압축 기법을 이용한 WSP의 기능 확장과 성능 평가

(Performance Evaluation of WSP with Capability Extension using Compression Techniques)

김 기 조 † 이 동 근 † 임 경 식 ††
(Kijo Kim) (Donggun Lee) (Kyungshik Lim)

요 약 Wireless Application Protocol(WAP) 포럼에서는 무선 환경의 특성에 적합하도록 HyperText Transfer Protocol(HTTP)를 수정 보완한 Wireless Session Protocol(WSP)를 제안하였다. WSP는 무선 구간에서의 프로토콜 성능을 상당부분 개선하였고, 무선 환경을 고려한 다양한 서비스를 제공하고 있다. 본 논문에서는 기존의 WSP에 프로토콜 메시지 압축 기능을 추가하여 기존 프로토콜의 성능을 더욱 개선하였다. 그리고, 기존의 WSP 구현물과 프로토콜 메시지 압축 기능을 추가한 WSP+ 구현물의 성능을 측정하고 비교하였다. 그 결과 본 논문에서 제안한 프로토콜 메시지 압축기능은 기존 WSP에 비하여 응답 트래픽을 약 45% 감소시켰다. 그리고, 비트 오류 발생률이 10^{-4}인 실험 환경에서 패킷 손실률과 트랜잭션당 시간 지연을 40% 이상 개선하였다. 이러한 측정 결과에서 알 수 있듯이, 프로토콜 메시지 압축 기능은 무선 구간에서 전달되는 메시지의 크기를 줄임으로써 패킷의 손실률을 낮추어 트랜잭션 계층에서의 메시지 재전송 횟수를 줄이고 트랜잭션당 시간 지연을 단축하였다.

키워드 : 무선 인터넷, 세션 프로토콜, 프로토콜 구현, 압축

Abstract Wireless Session Protocol(WSP) which was updated and supplemented based on HyperText Transfer Protocol(HTTP) was designed by Wireless Application Protocol(WAP) forum regarding the characteristics of wireless environment. WSP improved the performance in wireless network, and introduced various facilities considering wireless environment. In this paper, we more improve the performance of WSP adding protocol message compression capability; we call improved WSP protocol as WSP+. And, we analysis the performance of each protocol with WSP and WSP+ implementation. As a result of experiment, the capability which proposed in this paper reduced a response traffic about 45%. In 10^{-4} bit error rate, we also found the packet loss rate and time delay per transaction of WSP+ was improved over 40%. Finally, we found that the protocol message compression capability reduces message retransmission count in transaction layer and shorten the delay time per transaction by reducing a message size.

Key words : WAP, WSP, HTTP, Protocol implementation, co-rutine, compression

1. 서 론

기존 무선 인터넷에서 컨텐츠를 전송하기 위하여 사용중인 세션 프로토콜은 W3C의 HTTP과 WAP 포럼의 WSP가 있다[1,2]. 무선 인터넷 환경은 유선 환경에 비하여 낮은 대역폭과 제한된 컴퓨팅 능력, 한정된 전력과 높은 데이타 전송 지연 그리고, 불안정한 접속 등의 다양한 문제점이 있다. HTTP는 이러한 무선 환경에 대한 특별한 대응 방안이 없으나, WSP는 이러한 문제점들을 극복하기 위한 다양한 기능들을 제공한다. WSP는 무선 구간의 낮은 대역폭을 고려하여 프로토콜 헤더를 바이너리 형태로 인코딩하며, 불안정한 망 접속에 유연하게 대처하기 위한 세션 관리 메커니즘을 제공한다. 그리고, 무선 구간에서의 주요한 서비스인 푸시 서비스를 위하여 푸시 기능을 제공한다[3].

† 비 회 원 : 경북대학교 컴퓨터학과
kijo@ccmc.knu.ac.kr
leedg@ccmc.knu.ac.kr
†† 종신회원 : 경북대학교 컴퓨터학과 교수
kslim@knu.ac.kr
논문접수 : 2002년 1월 10일
심사완료 : 2002년 5월 17일

본 논문에서는 WSP 프로토콜 헤더와 컨텐츠를 포함한 프로토콜 메시지를 압축하는 기능을 추가하여 WSP의 성능을 개선하는 방법을 제안한다. 이 기능은 무선 구간에서 전달되는 메시지의 크기를 줄임으로써 패킷의 손실률을 낮추어 트랜잭션 계층에서의 메시지 재전송 횟수를 줄이고 트랜잭션당 시간 지연을 단축한다.

2. 관련 연구

WAP의 주요 컨텐츠인 WML및 WMLScript는 게이트웨이에서 바이트코드로 컴파일되어 사용하며, 기존 컨텐츠의 크기를 약 70% 감소시켜준다[4]. 그러나, 이렇게 인코딩된 컨텐츠는 gzip, compress와 같은 기존의 데이타 압축 알고리즘으로 압축할 경우 16% ~ 22%정도의 압축 효과를 더 보인다[5]. 특히, WAP에서 이미지 표현을 위한 형식인 WBMP는 gzip을 사용하여 190개의 예제 파일을 압축한 결과 약 50% 이상의 압축 효과가 있었다. 이러한 특징들 때문에 WAP 포럼에서는 Wireless Transaction Layer Security(WTLS)나 또는 다른 계층에서 압축 기능을 추가하는 것을 향후 연구 과제로 지정하고 있다[5,6].

본 논문에서는 서버 모델의 코루틴 구현 방식으로 구현된 WAP 구현물에 프로토콜 메시지 압축 기능을 추가하여 성능을 측정하였다. 코루틴 방식은 응용 수준의 프로세스로 계층 프로토콜을 구현하기 위한 서버 모델의 한가지 구현 방식이다. 서버 모델은 모듈화, 은닉화 및 명확한 인터페이스를 제공함으로써 프로토콜 공학적 관점에서는 이상적인 접근 방법이다. 서버 모델에서는 계층간 통신은 메시지를 이용하고 프로토콜 계층을 서버라는 실행 단위로 구성하며, 서버를 구현하는 방식에 따라 세가지로 구분 할 수 있다. 첫째는 프로세스 모델이다. 이 모델은 서버 모델에서 제시한 서버를 실제 운영체제가 제공하는 프로세스로 구현하는 방식이다. 두번째는 쓰레드 모델이다. 이 모델은 서버를 운영체제가 제공하는 쓰레드를 이용하여 구현하는 방식이다. 셋째는 코루틴 모델이다. 이 모델은 서버를 코루틴으로 구현하는 방식이다. 그러므로, 코루틴 모델에서는 위 두 모델과는 달리 운영체제의 스케쥴러가 아니라 프로그래머에 의해 동기점에서 수행되어야 할 코루틴이 결정된다. 코루틴이 서버의 역할을 하므로 프로그래머가 직접 이 코루틴을 스케쥴링해야 한다. 특히, 이 모델은 코루틴이 서버가 되므로 운영 체제에서 프로세스나 쓰레드를 지원하지 않는 경우에도 구현이 가능하다. 그러므로, 임베디드 운영 체제와 같은 제한된 처리능력을 가진 시스템에서 다중 계층 프로토콜 시스템을 구현하는데 적합하다[7,8,9]

3. WSP 계층에서의 프로토콜 메시지 압축 기능 제안 및 설계

3.1 프로토콜 메시지 압축 기능 개요

WAP에서 사용되는 컨텐츠들을 데이타 압축 알고리즘으로 압축할 경우 압축률이 높고, WAP 표준에서도 이러한 압축 기능을 권고하고 있으므로 WSP 계층에서 프로토콜 메시지를 압축하여 전달하는 기능을 제안한다. 본 논문에서는 컨텐츠 형식에 따른 압축 알고리즘을 협상하는 방법과 그에 따라 프로토콜 메시지를 압축하는 방법만을 제시하며, 각 컨텐츠 형식과 다양한 압축 알고리즘에 따른 압축률에 관한 측정은 본 논문의 영역을 벗어난다.

데이타 압축은 전송되는 패킷의 크기를 줄여주는 역할을 하지만, 데이타를 압축하거나 압축 해제 할 때 시간 지연이 발생한다. 그러므로, 메시지 재전송 기능이 있는 Wireless Transaction Protocol(WTP) 계층은 적합하지 않다. 그리고, Wireless Datagram Protocol (WDP)나 WTLS 계층에서 메시지를 압축할 경우에는 WTP의 동작을 방해하는 것 이외에도 암호화로 인하여 압축률도 현저히 낮으므로, WSP 계층에서 프로토콜 메시지를 압축하는 것이 적합하다.

이러한 WSP의 프로토콜 메시지 압축 기능은 WSP 계층에서 만들어진 프로토콜 메시지를 압축하여 전달함으로써 메시지의 크기를 줄인다. WSP의 프로토콜 메시지는 크게 HTTP 헤더를 인코딩한 프로토콜 헤더 부분과 컨텐츠를 포함한 프로토콜 바디 부분으로 나뉜다. 프로토콜 메시지 압축 기능은 컨텐츠를 가장 잘 압축할 수 있는 알고리즘을 사용하지만 부가적으로 인코딩 된 헤더를 재압축하는 효과도 있다. 이때 사용되는 알고리즘은 컨텐츠의 형식에 따라 달라진다. 그리고, 프로토콜 메시지의 압축은 응답 메시지만 적용한다. 그 이유는 요구 메시지의 경우는 일반적으로 컨텐츠를 포함하지 않고, 메시지의 크기도 작으므로 압축률이 낮기 때문이다.

이 기능의 사용 여부와 컨텐츠에 따른 압축 알고리즘은 WSP 프로토콜 기능 협상 시에 결정한다. 프로토콜 메시지 압축 기능을 위한 프로토콜 기능 협상 방식은 기존의 WSP 프로토콜 기능 협상인 단방향 협상 방식을 따른다. 즉, 클라이언트는 사용 가능한 압축 알고리즘 리스트를 서버로 전달하고, 서버는 알고리즘 리스트에서 서버가 수용할 수 있는 압축 알고리즘 리스트를 선택하여 응답한다. 그리고, 새롭게 정의되는 컨텐츠 형식에 대한 정보나 각 컨텐츠에 따른 압축 알고리즘에 대한 정보를 서버에서 관리하여, 버전 관리나 새로운 컨

텐츠 형식의 추가 등의 변화에 유동적으로 대응한다.

기존의 HTTP와 WSP도 응답 컨텐츠를 압축하는 기능을 제공한다. 이 기능은 클라이언트에서 'Accept-Encoding' 헤더를 사용하여 압축 해제가 가능한 압축 알고리즘을 서버에 알려주면, 서버는 'Content-Coding' 헤더를 사용하여 컨텐츠를 압축한 알고리즘을 명시함으로써 이루어진다[10]. 이 방법은 트랜잭션마다 이러한 헤더 정보들을 부가적으로 사용하여야 하며, 응용 계층에서 압축 해제하기 위한 기능을 제공해야하는 등 응용 개발을 복잡하게 한다. 그리고, 전체 프로토콜 메시지를 구성하는 비율이 높은 프로토콜 헤더 부분을 재압축하는 기능도 없다.

그러나, 본 논문에서 제시하는 프로토콜 메시지 압축 기능은, 세션 설정 시에 압축 방식을 교환하고 세션에서 일관성 있게 적용함으로써, 응용 계층에서는 하부 계층의 복잡한 동작 과정을 고려할 필요 없이 투명하게 성능 개선 효과를 획득하게 된다. 이 기능은 낮은 무선 구간의 대역폭을 효과적으로 사용할 수 있을 뿐 아니라, 무선 구간의 높은 오류 발생률에 따른 패킷 손실을 줄여줌으로써 컨텐츠의 평균 응답 시간도 단축시킨다.

3.2 프로토콜 기능 협상 방법

프로토콜 메시지 압축 기능을 사용하기 위한 프로토콜 기능 협상 방법은 기존의 WSP의 협상 방식인 단방향 기능 협상 방식을 따른다. WSP에서 제시하는 단방향 기능 협상 절차는 다음과 같다.

- 클라이언트 사용자는 기능 협상 값들을 제안한다.
- 클라이언트 프로토콜은 사용자가 제안한 기능 협상 값들을 지원 할 수 있는지에 따라 제안한 기능 협상 값들을 수정한다.
- 서버 프로토콜은 클라이언트가 제안한 협상 값을 서버 프로토콜이 지원할 수 있는 기능을 고려하여 수정한다.
- 서버 사용자는 수정된 협상 값들을 고려하여 현재 세션에서 사용될 협상 값을 응답한다. 응답 값이 생략되었을 경우는 클라이언트가 제안한 값을 서버가 그대로 사용함을 의미한다.
- 클라이언트는 서버가 선택한 기능 협상 값을 알게 된다. 또한, 이렇게 협상된 값들은 새로운 기능 협상이 이루어지기 전까지 기본 설정 값이 된다.

프로토콜 메시지 압축 기능을 제공하기 위하여 기존의 프로토콜 기능 협상 방법에 프로토콜 옵션과 컨텐츠에 따른 압축 알고리즘 정보를 교환하기 위한 메커니즘이 추가되었다. 기존의 프로토콜 옵션은 세션에서 사용할 프로토콜의 기능을 지정하는 역할을 한다. 현재 WSP의 프로토콜 옵션에서 정의된 값들은 푸시, 세션 재개 기능과 응답 헤더의 사용 여부에 대하여 정의하고 있다. 본 논문에서는 현재 세션에서 프로토콜 메시지 압축 기능의 사용 여부를 결정하기 위한 값으로 0x08을 새로 정의한다.

길이	식별자	압축 알고리즘
uintvar	multiple octets	multiple octets

그림 1 압축 기능 협상 정보 인코딩 형식 - 요구시

길이	식별자	압축 알고리즘	바이너리 정의 값
uintvar	multiple octets	multiple octets	octet

그림 2 프로토콜 메시지 압축 기능 협상 정보 인코딩 형식 - 응답시

압축 기능 협상 정보 인코딩 형식은 컨텐츠 형식에 따른 압축 알고리즘에 대한 정보를 교환하기 위하여 사용한다. 그림 1, 그림 2는 압축 기능을 위한 협상 정보 인코딩 형식이다. 프로토콜 메시지 압축 기능 인코딩 형식임을 명시하기 위한 식별자는 0x08로 정의한다. 클라이언트에서는 협상 시에 압축 해제가 가능한 모든 압축 알고리즘 리스트를 서버에 전달한다. 압축 알고리즘의 기술 방식은 WSP 헤더 인코딩 기법과 동일하다. 즉, 바이너리 정의 값이 있을 경우에는 그 값을 사용하며, 정의 값이 없을 경우에는 '\0'를 포함한 텍스트가 사용된다. 현재 정의된 압축 방식의 정의 값은 gzip의 경우 0x00, compress의 경우 0x01, zip의 경우 0x02, deflate의 경우 0x03이다.

서버는 클라이언트가 지원하는 압축 알고리즘 리스트에서 서버가 수용할 수 있는 압축 알고리즘을 선택하여 그림 2와 같이 인코딩하여 응답한다. 이때 바이너리 값이 정의되지 않은 압축 알고리즘은 0x80 ~ 0xF0 범위의 값들 중에서 임의의 값을 동적으로 정의하여 그 세션 기간 동안 사용한다. 서버가 동적으로 바이너리 값을 할당한 경우에는 '\0'를 포함한 텍스트 토큰과 그에 대한 바이너리 정의 값을 모두 전달한다. 만약 서버가 응답 정보를 생략하게 되면 클라이언트는 암시적으로 제안한 알고리즘 리스트가 허용됨을 알게된다. 그러나, 서버가 수용할 수 없는 압축 방식의 경우에는 바이너리 정의 값 필드에 0xFF 값을 설정하여 클라이언트에 명시적으로 알려준다. 그리고, 프로토콜 옵션에 프로토콜 메시지 압축 기능은 활성화시키고, 구체적인 압축 알고

리즘을 명시하지 않은 경우에는 기본 압축 알고리즘인 gzip을 사용한다[11].

3.3 설계

프로토콜 메시지 압축 기능은 기존의 WSP에 응답 프로토콜 메시지를 압축하는 기능을 추가한 것이다. 이 기능은 전체 메시지의 크기를 줄여줌으로써, 하부 계층에서의 패킷 손실을 줄이고 컨텐츠 전송 지연 시간을 단축한다. 이 기능의 사용 여부와 압축 방식은 프로토콜 기능 협상 시에 결정된다. 본 단락에서는 프로토콜 기능 협상 과정에서 교환된 정보를 이용하여 프로토콜 메시지를 압축하기 위한 방법에 대하여 기술한다.

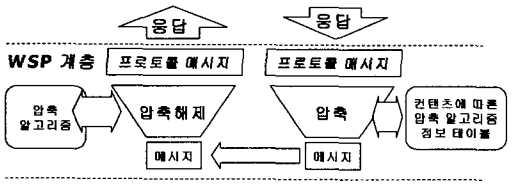


그림 3 WSP 프로토콜 메시지 압축 기능 동작 과정

압축 알고리즘	길 이	압축된 WSP 프로토콜 메시지
octet	uintvar	multiple octets

그림 4 프로토콜 메시지 압축 기능을 적용시의 프로토콜 메시지 형식

그림 3은 WSP 프로토콜 메시지가 압축 및 압축 해제되는 동작 과정을 개략적으로 나타낸다. WSP 관리 영역에는 프로토콜 기능 협상 과정에서 협상된 알고리즘 리스트에 대한 정보가 있다. 특히, 서버에서는 알고리즘 리스트 이외에도 컨텐츠에 따라 적용할 압축 알고리즘에 대한 테이블도 유지된다. 클라이언트의 요구 메시지는 압축 없이 전달하며, 서버는 정상적인 메시지 처리 과정에 의하여 만들어진 응답 프로토콜 메시지를 압축하여 전달한다. 이때, 서버는 협상된 압축 알고리즘 중에서 응답 메시지에 포함된 컨텐츠를 가장 잘 압축할 수 있는 압축 알고리즘을 선택하여 압축한다. 그리고, 압축되어 전달된 프로토콜 메시지는 클라이언트에서 압축 해제됨으로써 원본 프로토콜 메시지가 복원되어 기존의 WSP 메시지 처리 루틴에 따라서 처리된다.

프로토콜 메시지 압축 기능을 적용할 경우의 프로토콜 메시지의 형식은 그림 4와 같다. 프로토콜 메시지는 압축 알고리즘을 명시하는 필드와 압축된 프로토콜 메시지의 길이 그리고, 압축된 WSP 프로토콜 메시지 페이로드 필드로 구성된다. "압축 알고리즘" 필드는 프로토콜 기능 협상 시에 교환된 압축 알고리즘에 대한 바이너리 값이 사용된다. 이 필드에는 서버에서 유지하는 컨텐츠에 따른 압축 알고리즘 테이블을 기준으로 설정된다. 그리고, 서버는 컨텐츠 압축률이 높지 않을 경우 압축을 하지 않는 기능도 제공한다. 프로토콜 메시지를 압축하지 않을 경우에는 압축 알고리즘을 0x7F로 정의한다. 그 외에 압축 방식 테이블에 정의되지 않은 컨텐츠 형식의 경우에는 안정적인 데이타 압축률이 검증된 gzip을 사용한다[11]. gzip은 기본 압축 알고리즘이므로 프로토콜 메시지 압축 기능을 사용하기 위해서는 기본으로 지원되어야 한다. '길이' 필드는 'uintvar'이라는 WSP에서 정의한 데이타 형식을 사용한다. 이 데이타 형식은 인코딩 할 값이 커짐에 따라 값을 표현하기 위한 변수의 크기가 커지는 특징을 가진다. 즉, 작은 값은 하나의 옥텟으로 표현 하지만, 값이 커질수록 여러 개의 옥텟을 사용하여 값을 표현한다. 'uintvar'의 첫 번째 비트는 "Continue bit"이며 이후에 연결될 옥텟이 있을 경우는 '1'로 없을 경우에는 '0'으로 설정한다. 그 이외에 일곱 비트는 페이로드 영역으로 실제 값이 저장된다. 그리고, "압축된 WSP 프로토콜 메시지" 필드에서는 압축 알고리즘 테이블에 따라서 압축된 WSP 프로토콜 메시지를 저장하는 부분이다.

4. 구 현

프로토콜 메시지 압축 기능은 기존의 WAP 구현 환경을 기반으로 프로토콜 메시지 압축 기능을 추가하게 되므로, 기존의 WAP 구현 모델을 설명하고 프로토콜 메시지 압축 기능을 추가하기 위한 방법을 기술한다.

4.1 WSP 구현 모델

WAP 프로토콜은 WAE, WSP, WTP, WTLS 와 WDP/UDP 프로토콜 스택으로 구성되어 있다[2]. WAP 프로토콜을 개발하기 위한 플랫폼은 리눅스 커널

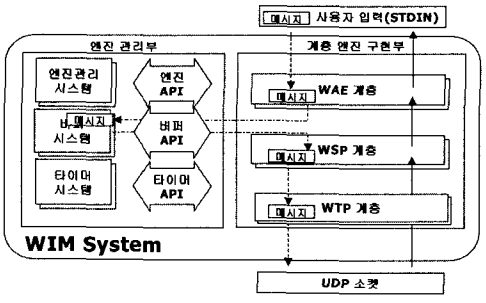


그림 5 단일쓰레드용 WAP 구현 모델 구조 설계

```
void main()
{
        engine_mgr_init();                  //엔진 관리 영역 초기화
        timer_init();           //타이머 초기화
        buf_sys_init();                     //버퍼 시스템 초기화
        engine_register(wdp_engine, wtp_engine,
                        wsp_engine, wae_engine);        //관리 영역에 각 프로토콜 계층 등록
        wdp_init(), wtp_init(), wsp_init(), wae_engine() ;//각 프로토콜 엔진 초기화
        while(1){
        ret = event_dispatch(msgq);                     //메시지 발생 큐와 처리할 계층 정보 리턴
        switch(ret){
                case WAE: wae_engine(msgq), break; //WAE 엔진 모듈
                case WSP: wsp_engine(msgq), break; //WSP 엔진 모듈
                case WTP: wtp_engine(msgq), break; //WTP 엔진 모듈
                case WDP: wdp_engine(msgq), break; //WDP 엔진 모듈
        }//end of switch
        }//end of while
}
```

그림 6 엔진 관리 시스템 수도 코드

2.2.14이고 UDP를 이용하여 개발한다. 프로토콜을 개발하기 위한 모델로서는 프로그래밍의 복잡성은 증가하지만 단일 쓰레드 형태로 수행 될 수 있으며, 메시지 전달 방식으로 계층간 통신을 함으로써 모듈화, 은닉화 및 명확한 인터페이스를 제공하는 모델인 서버 모델의 코루틴 모델을 기본으로 한다. 그림 5의 WAP 구현 모델(WAP Implementation Model:WIM)은 WAP 프로토콜 스택을 구현하기 위한 모델을 간략히 나타낸 것이다. WAP 구현 모델은 엔진 관리부과 계층 엔진 구현부로 나뉜다.

엔진 관리부에서는 메시지 재전송과 같은 작업을 수행하기 위한 타이머 시스템, 프로토콜 계층간 메시지 전달을 위한 버퍼 시스템, 계층 엔진의 호출과 스케쥴링의 역할을 담당하는 엔진 관리 시스템이 있다. 각 시스템들의 기능은 라이브러리 형태로 만들어지며, 프로토콜 계층 구현부에 API를 제공한다. 계층 엔진 구현부는 WSP/WTP 등의 구현 부분인 '엔진'을 구현하는 부분이다. 각 계층간의 인터페이스는 프리미티브로 결정되며, 내부 수행 동작은 프로토콜의 동작을 기술한 WSP 표준의 "상태 테이블"을 기반으로 구현된다. 프로토콜 계층간 인터페이스인 프리미티브는 버퍼 시스템에서 정의하는 버퍼 메시지로 구조화되어, 버퍼 시스템 API를 통하여 다음 계층에 전달된다. 각각의 계층 프로토콜 엔진은 타이머 시스템, 엔진 관리 시스템 및 버퍼 시스템 라이브러리가 제공하는 API들을 사용하여 구현된다. 이렇게 구현된 엔진들은 엔진 관리 시스템에 등록되어 수행된다. 그림 6은 엔진 관리 시스템의 핵심 부분을 수도 코드화 한 것이다.

그림 7은 WSP 엔진의 구조를 간략히 나타낸 것이다. 엔진 관리 시스템에 의하여 호출된 WSP 엔진은 입력된 메시지를 처리함으로써 프로토콜을 동작하게 한다. 프로토콜의 동작은 WSP 표준에 기술된 프리미티브와 상태 테이블을 기반으로 구현하며, 프로토콜 메시지 압축 기능 추가 시에 프리미티브와 상태 테이블은 수정되지 않는다. 상태 테이블은 상·하위 계층 및 상응 계층과의 상호 동작을 표현한 것이므로, 상·하위 계층의 프리미티브 이벤트를 기반으로 프로토콜의 동작을 기술하고 있다. WSP 엔진 모듈의 구성은 먼저 상태 테이블에 기술된 '상태' 정보에 따라서 분기하고, 분기된 모듈에서 다시 "프리미티브 이벤트"에 따라서 분기된다. 마지막으로 분기된 모듈에서는 상태 테이블의 "작업 필드"에 기술된 내용에 따라 입력된 메시지를 처리한다.

WSP 엔진의 시작 모듈인 wsp_engine은 상위 계층

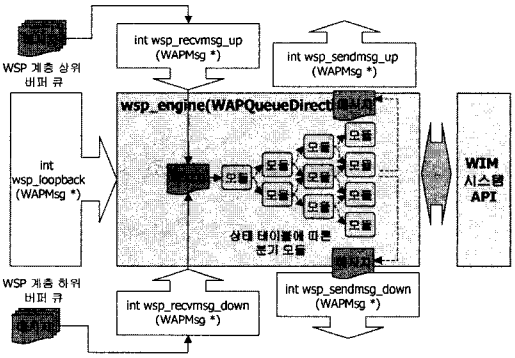


그림 7 WSP 엔진 프로토타입 및 구조

과 하위 계층으로부터 전달되는 메시지를 버퍼링하기 위하여 버퍼 시스템에 상·하위 버퍼 큐가 있다. WSP 계층 버퍼 큐에 메시지 입력이 있을 경우 wsp_engine 이 호출되며, 엔진 내부에서는 버퍼 메시지가 입력된 버퍼 큐를 식별하여 메시지를 출력한다. 상위 큐로부터 메시지를 읽을 때는 wsp_recvmsg_up를 사용하고, 하위 큐로부터 메시지를 읽을 때는 wsp_recvmsg_down을 사용한다. 그리고, 프로토콜 동작 과정 중에 상위 또는 하위 계층으로 메시지를 전달할 경우에 사용하는 함수로는 wsp_sendmsg_up, wsp_sendmsg_down이 있다. 또한 프로토콜 수행 중에 현재 계층으로 메시지를 다시 전달하는 경우에는 wsp_loopback이 사용된다. 이 함수들은 WSP를 구현하기 위하여 버퍼 시스템이 제공하는 API들이다. WSP 엔진 내부에서는 이러한 함수들을 이용하여 상·하위 계층으로 메시지를 전달하며, 다른 계층 엔진에서도 이와 유사한 형태의 함수들을 이용하여 동일한 방법으로 상응 계층에 메시지를 전달한다. 이렇게 전달된 메시지들은 WSP의 상태 테이블에 따라 구현된 분기 모듈들에 의하여 처리된다.

4.2 프로토콜 메시지 압축 기능 구현

프로토콜 메시지 압축 기능은 서버의 컨텐츠 응답 메시지만을 압축하므로 기존의 wsp_sendmsg_down과 wsp_recvmsg_down 함수를 이용하여 서버 측에서는 wsp_comp_sendmsg_down 모듈을, 클라이언트 측에서는 wsp_comp_recvmsg_down 모듈을 새롭게 추가하여 프로토콜 메시지 압축 기능을 구현한다. 서버의 wsp_comp_sendmsg_down에서는 관리 영역에 기술된 컨텐츠에 따른 압축 알고리즘 테이블을 참조하여 WSP 프로토콜 메시지를 압축하여 전달한다. 그리고, 클라이언트의 wsp_comp_recvmsg_down은 버퍼 큐로부터 메시지를 출력하고 분석하여 압축을 해제하는 역할을 한다. 프로토콜 메시지 압축 기능은 이 두 함수를 수정하여 구현하기 때문에 기존의 wsp_engine 내부 모듈의 수정을 최소화 할 수 있으며, 압축 해제된 WSP 프로토콜 메시지는 기존의 프로토콜 처리 루틴을 동일하게 사용한다. 다음 그림 9, 그림 10은 클라이언트와 서버에서 수정되는 두 모듈의 수도 코드이다.

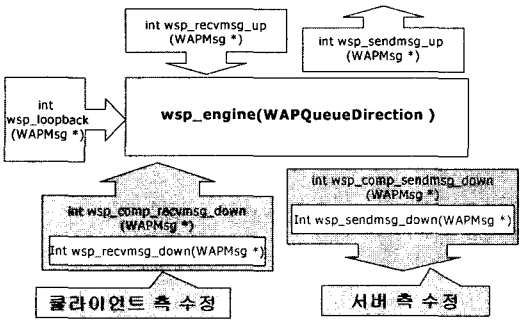


그림 8 프로토콜 메시지 압축 기능 추가 시 수정된 부분

```
wsp_comp_sendmsg_down(WAPMsg *WSP메시지){
   중략
   if(연결형 서비스&&프로토콜 메시지 압축기능 사용)){
      압축 알고리즘 리스트로부터 컨텐츠에 따른 압축
      알고리즘 선택;
      선택된 압축 알고리즘에 따라 메시지 압축;
      압축 메시지 구성;
      wsp_sendmsg_down(압축 메시지); //하부 계층으로 메시지 전달;
   }
   하략
}
```

그림 9 서버의 wsp_comp_sendmsg_down 수도 코드

```
wsp_recvmsg_down(WAPMsg *WSP메시지){
   중략
   if(연결형 서비스&&프로토콜 메시지 압축기능 사용)){
      wsp_recvmsg_down(메시지); //하부 계층에으로부터 메시지 전달;
      압축 메시지로부터 압축 알고리즘 추출;
      알고리즘에 따라 메시지 압축 해제;
      원본 프로토콜 메시지 추출;
      WSP메시지 = 원본 프로토콜 메시지;
                  //기존 WSP 프로토콜 메시지 처리 루틴에 메시지 전달;
   }
   하략
}
```

그림 10 클라이언트의 wsp_comp_recvmsg_down 수도코드

5. 성능 평가

본 장에서는 압축 방식에 따른 무선 인테넷 용 컨텐츠의 압축률과, 각 프로토콜들의 패킷 손실 및 전송 시간 지연을 측정한다. 컨텐츠 압축률은 본 논문에서 제안한 WSP 프로토콜 메시지 압축 기능의 제안 근거가 되며, 각 프로토콜의 패킷 손실과 전송 시간 지연의 차이를 설명하기 위한 자료가 된다.

5.1 컨텐츠 압축률 분석

측정 대상은 압축 알고리즘에 따른 표본 컨텐츠의 압축률과 압축 시에 소요된 CPU 사용량이다. 압축 알고리즘은 텍스트 압축에 많이 사용되는 gzip, compress, zip을 사용하였고, 표본 컨텐츠는 WML 바이트코드,

WMLScript 바이트코드와 WBMP이다. 압축률을 계산하는 방법은 그림 11과 같다.

CPU 사용량은 압축할 때 발생하는 전력 소비와 시간 지연의 정도를 알아보기 위하여 측정한다. 높은 CPU 사용량은 패킷의 전송 지연으로 WTP의 성능을 떨어뜨릴 수 있으며, 단말에서 많은 전력을 소비할 수 있다. CPU 사용량은 시스템 시간을 밀리 초 단위로 측정하며, 압축한 직후의 시간에서 압축하기 직전의 시간을 감산한 값으로 정한다. 이때 사용된 시스템 함수는 마이크로 초 단위의 정밀성을 가지는 gettimeofday 함수를 사용하였다. 컨텐츠의 압축 시에는 system 함수를 사용하였으며, CPU 사용량을 측정하기 위한 수도 코드는 그림 12와 같다.

$$\text{압축률}(\%) = 100 - \frac{\alpha}{\beta} \times 100$$

α = 압축 후 파일 크기 평균
β = 압축 전 파일 크기 평균

그림 11 압축률 식

```
static      struct timeval  m_last, m_cur;
long        msec1, msec2, diffmsec;

gettimeofday(&m_last, NULL);
system("gzip fil1.wml");            //압축 수행
gettimeofday(&m_cur, NULL);
diff.tv_sec             = m_cur.tv_sec - m_last.tv_sec;
diff.tv_usec            = m_cur.tv_usec - m_last.tv_usec;
msec1       = diff.tv_sec * 1000000;
msec2               = diff.tv_usec;
diffmsec    = msec1 + msec2; //마이크로 초 단위로 계산
```

그림 12 CPU 시간 측정 모듈

측정에 사용된 WML 바이트코드의 표본 개수는 308개이며 WMLScript 바이트코드는 43개 그리고, WBMP의 표본 개수는 197개이다. 측정한 시스템의 운영 체제는 리눅스 커널 2.2.16이며, 사양은 펜티엄 II MMX 350Mhz, RAM 190M이다. 이러한 환경에서 측정된 결과는 표 1 ~ 표 3과 같다. 압축 알고리즘에 따른 컨텐츠의 크기와 CPU 사용량은 평균값이다.

표 1 WMLC 압축 결과(WMLC 원본 평균 크기는 374 바이트)

목록 / 압축 알고리즘	크기(Byte)	압축률(%)	CPU사용량 (밀리 초)
gzip	264	30	11.38
zip	379	-1	17.40
compress	286	24	14.71

표 2 WMLSC 압축 결과(WMLSC의 평균 크기는 195 바이트)

목록 / 압축 알고리즘	크기(Byte)	압축률(%)	CPU사용량 (밀리 초)
gzip	188	4	10.92
zip	304	-55	11.86
compress	176	10	13.97

표 3 WBMP 압축 결과(WBMP의 평균 크기는 801 바이트)

목록 / 압축 알고리즘	크기(Byte)	압축률(%)	CPU사용량 (밀리 초)
gzip	279	66	11.24
zip	394	51	21.92
compress	311	41	14.19

표 1은 WML의 바이트코드인 WMLC 파일을 압축한 결과를 나타낸 표이다. 측정 결과에서는 gzip 방식이 약 30%의 압축 효과가 있었으며, CPU 사용 시간도 15 밀리 초 이하로 측정되어 부하가 크지 않았다. 표 2는 WMLScript의 바이트코드인 WMLSC 파일을 압축한 결과를 나타낸 표이다. WMLSC의 경우에는 모든 알고리즘들이 10% 이하의 낮은 압축률을 기록하였다. 이는 WMLSC가 텍스트 정보를 많이 포함하고 있지 않기 때문이다. 그러나, CPU 사용 시간은 모두 15 밀리 초 이하로 WMLC의 경우처럼 많지 않았다. 표 3은 WBMP를 압축한 결과를 보여주며 gzip, zip 그리고 compress 방식 모두 40% 이상의 높은 압축률을 보였다. 그 중에서 gzip이 가장 높은 압축률과 가장 낮은 CPU 사용량을 기록하였다.

그러므로, 컨텐츠를 포함하는 프로토콜 메시지를 gzip, compress 등의 압축 알고리즘으로 압축하게되면 프로토콜 메시지를 구성하고 있는 컨텐츠와 프로토콜 헤더가 모두 압축되므로 프로토콜 메시지 크기를 크게 줄여준다. 프로토콜 메시지를 압축할 경우에는 프로토콜 메시지에 포함된 컨텐츠에 따라서 압축 방식을 적용하게 되므로 표 1, 표 2, 표 3에서 제시한 압축 방식에 따른 압축률은 압축 방식 테이블을 구성하는데 사용된다. 즉 WMLC와 WBMP를 압축할 때는 gzip을 사용하고 WMLSC의 경우는 compress를 사용하는 것이 효과적이다. 그러나, 이 결과는 세 가지 알고리즘 만을 고려한 것이므로 향후 다양한 압축 알고리즘과 컨텐츠에 따른 압축률에 대한 연구가 필요하다.

기존의 WAP에서는 WML이나 WMLScript의 크기

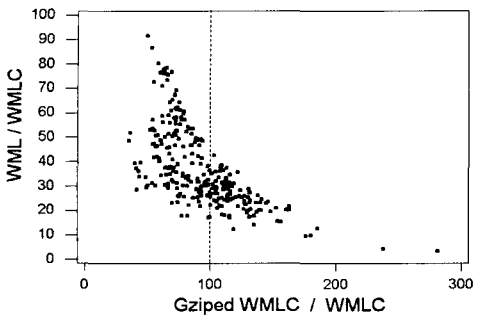


그림 13 WML 압축 비율표

를 줄이기 위하여 게이트웨이에서 컴파일을 수행하며, 그 방법은 관용 토큰을 바이너리로 맵핑하는 방식이다. 그러나, 이러한 방식은 컨텐츠 내에 텍스트가 많을 경우에는 압축률이 낮아지게 되므로, 다른 형태의 압축 방식을 적용하여 높은 압축률을 획득 할 수 있음을 의미한다. 그림 9는 이러한 압축에 따른 상관 관계를 잘 보여주는 예이다. 그림 13은 WML 컴파일러에 의한 압축률이 낮으면 gzip에 의한 압축률이 높으며, WML 컴파일러에 의한 압축률이 높으면 gzip에 의한 압축률이 낮음을 보여주고 있다. 그러나, WMLScript의 경우는 텍스트를 포함하는 비율이 낮기 때문에 이러한 현상을 잘 보여 주지 못하며 압축률도 낮다.

5.2 패킷 손실 및 시간 지연 측정

패킷의 손실은 패킷의 크기에 따라 발생 할 확률이 높으며, 패킷의 손실이 많아지게 되면 패킷 재전송으로 인한 트랜잭션의 시간 지연이 길어지게 된다. 그러므로, 본 실험에서는 각 프로토콜의 성능을 비교하기 위하여 예제 트랜잭션을 수행할 때 발생하는 트래픽의 총량과 트랜잭션 당 사용되는 평균 패킷 크기를 측정하고자 한다.

측정을 위한 트랜잭션 회수는 133번이며, 트랜잭션을 위한 표본 메시지는 133개의 HTTP 요구/응답 메시지로 구성되어 있다. WML 파일을 요청하는 표본이 79개, WBMP 파일을 요청하는 표본이 38개 그리고 WML Script를 요청하는 표본이 16개이다. 응답 표본 중에서 WML과 WMLSCript의 경우에는 바이너리 형식으로 컴파일 되어있다. 하부 전송 계층 프로토콜은 WSP/WTP를 사용한다. 테스트를 위하여 HTTP 메시지를 직접 전달하기 위한 기능을 WSP 프로토콜에 추가하였으며, 메소드 형식 WSP_HTTP_GET을 사용한다. 압축 알고리즘은 컨텐츠 형식에 관계없이 기본 설정 값인 gzip을 사용한다. 테스트 프로토콜은 4가지이며 다음과 같다.

- HTTP 요구/응답 - HTTP
- 프로토콜 메시지 압축 기능을 적용한 HTTP 요구/응답 - HTTP+
- WSP 요구/응답 - WSP
- 프로토콜 메시지 압축 기능을 적용한 WSP 요구/응답 - WSP+

표 4 각 프로토콜에 따른 요구/응답 트래픽 측정

항 목 / 프로토콜	평균 메시지	
	평균 요구(Byte)	평균 응답(Byte)
HTTP	238	1015
HTTP+	238	570
WSP	109	822
WSP+	109	444

표 4는 각 프로토콜을 이용하여 표본 트랜잭션을 수행할 결과로 측정된 요구/응답 메시지의 평균 크기를 측정한 결과이다. 요구 메시지 크기는 WSP의 헤더 인코딩으로 WSP가 HTTP보다 작았다. 응답 메시지의 경우는 압축 기능을 추가한 HTTP+와 WSP+의 경우가 작았다. 특히, 헤더 인코딩으로 인한 패킷 크기의 감소율 보다 프로토콜 메시지 압축에 의한 패킷 크기 감소율이 더욱 높게 측정되었다. 이는 프로토콜 메시지 압축 기능이 뛰어난 성능을 발휘함을 증명한다. 전달되는 평균 메시지의 크기가 줄어들게 되면 전송되는 패킷이 손실될 확률이 줄어 패킷 재전송이 줄어들고, 패킷 전송 지연 시간이 단축된다. 특히, 무선 TCP나 WTP의 경우처럼 무선 구간에서 사용되는 전송 프로토콜들은 무선 구간의 시간 지연을 고려하여 패킷 손실 시에 재전송하는 시간 간격을 크게 설정하므로, 패킷 재전송으로 인한 지연 시간은 유선 구간에서보다 더욱 커지게 된다.

다음은 각 프로토콜의 트랜잭션당 패킷 손실 개수 및 지연 시간을 측정하고자 한다. 측정하는 방법은 그림 14와 같은 성능 측정 모델을 기반으로 하며, 표본 트랜잭션은 요구/응답 트래픽 측정시의 표본을 그대로 사용한다. WTP의 메시지 재전송 시간은 무선 구간의 특성을 반영하여 1초로하고, 대역폭을 64 Kbps로 설정하였으며 비트 오류 발생률을 변동시키면서, 패킷 손실과 트랜잭션당 시간 지연을 각 프로토콜의 종류에 따라 구분하여 측정한다. 대역폭은 실제 인터넷 환경을 시뮬레이션하기 위한 도구인 레드콤(Radcom)의 인터넷 시뮬레이터를 이용하여 시뮬레이션하고, 비트 오류는 옵넷(OPNET)에서 제공하는 PPP의 BER 생성 모듈을 사용하여 발생시킨다. 일반적인 무선 구간에서의 비트 오류 발생률이

10^{-3}에서 10^{-4}이므로 본 실험에서의 비트 오류 발생율을 10^{-4}에서 10^{-6} 범위로 하여 측정하였다[12,13]. 측정 시스템은 클라이언트/서버 모두 리눅스 커널 2.2.16이며, 사양은 펜티엄 II MMX 350Mhz, RAM 190M 이다. 이러한 환경을 기반으로 측정된 값은 그림 15과 그림 16과 같다.

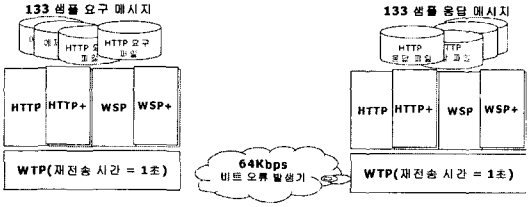


그림 14 프로토콜 성능 측정 모델

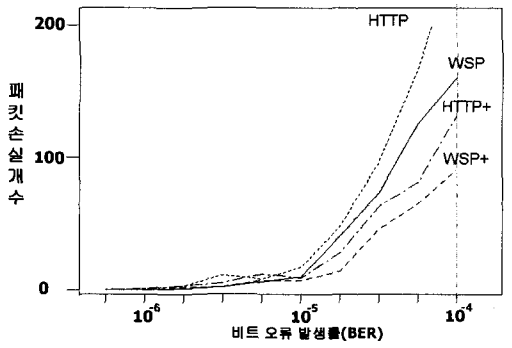


그림 15 비트 오류 발생률에 따른 패킷 손실 개수

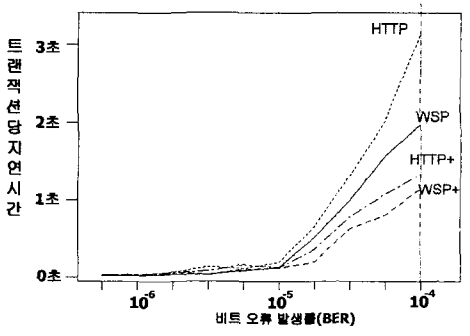


그림 16 비트 오류 발생률에 따른 트랜잭션당 지연 시간

그림에서 나타난 바와 같이 오류 발생률이 높아질수록 WSP+, HTTP+, WSP, HTTP의 순서로 패킷 손실률이 낮았다. 트랜잭션당 평균 전송 시간 지연도 패킷 손실과 동일한 순서로 작았다. 위 실험 결과는 무선 환경에서 WSP를 이용한 컨텐츠 전달 방법이 HTTP보다 패킷 손실률과 지연 시간이 작아서 무선 구간에 더 적합함을 보여준다. 그리고, 그림 16에서 비트 오류 발생률이 10^{-4}일 경우를 기준으로 보면 기존 프로토콜에 압축기능을 추가한 WSP+와 HTTP+가 기존 프로토콜들 보다 트랜잭션 당 응답 시간이 1/2정도로 단축되었음을 알 수 있다. 이는 프로토콜 메시지 압축 기능이 기존의 HTTP와 WSP의 성능을 상당부분 향상 시켰음을 보여주며, WSP+의 성능이 가장 우수함을 보여준다.

6. 결론 및 향후 연구

WSP는 기존의 HTTP보다 실험 환경에서 낮은 패킷 손실률과 평균 시간 지연을 보였다. 그리고, 본 논문에서 제안한 WSP 프로토콜 메시지 압축기능은 기존 WSP의 응답 트래픽을 약 45% 감소시켰고, 기존의 WSP보다 상당히 낮은 패킷 손실과 시간 지연을 보였다. 비트 오류 발생률이 10^{-4}인 실험 환경에서의 WSP의 패킷 손실 개수는 약 160개 였으며, WSP+의 경우는 약 90 개로 40% 이상의 성능 향상을 보였다. 그리고, 같은 환경에서 트랜잭션당 시간 지연의 측정 결과에서는 WSP의 경우는 약 2000 밀리 초이었으며, WSP+의 경우는 1200 밀리 초로 약 40% 단축되었다. 특히, 압축 기능은 HTTP에서도 유사하게 성능 향상을 보여, 압축 기능을 추가한 HTTP가 대역폭 절감 측면에서 WSP보다 우수하였다. 즉, WSP와 WSP+는 단말의 성능이 열악하며 대역폭이 낮은 무선 구간에서 뛰어난 성능을 발휘한다.

그러나, 향후 상용화 될 IMT-2000 망은 현재 보다 향상된 기능을 가지는 단말들이 사용되며, 무선 구간의 대역폭도 영상을 전송할 정도로 높아진다. 그러므로, WAP 포럼에서는 앞으로 상용화될 IMT-2000 망에서의 무선 인터넷 환경을 구축하기 위한 목적으로 HTTP/TCP/IP 프로토콜 스택을 포함한 WAP 2.0 프로토콜 스택 구조를 새롭게 정의하였다[14,15]. 그렇지만, WAP 2.0이 실현되기 위한 인프라인 IMT-2000 망이 일반화되기 위해서는 많은 시간이 필요하며, IMT-2000 서비스가 개시되더라도 상당 기간은 기존의 WAP 1.2를 지원하는 단말들이 계속 사용될 것으로 예상되기 때문에 현재 WAP 2.0에서는 WAP 1.2와 WAP 2.0를 동시에 지원하는 듀얼 스택을 권고하고 있다. 그러므로, 고속 무선 인터넷망이 일반화되더라도, 낮은 대역폭과 제한된 단말 성능을 가지는 환경을 기반으로 하는 응용에서는 WSP와 WSP+ 프로토콜이 좋은 참조

모델이 될 것이다[14,16,17].

향후 프로토콜 메시지 압축 기능을 적용할 경우 컨텐츠에 따른 압축 알고리즘 테이블을 구성하기 위한 정보가 부족하므로, 다양한 컨텐츠와 알고리즘을 이용한 압축률 측정 결과가 필요하다. 그러므로, 각 컨텐츠에 따라 가장 높은 압축률을 가지는 압축 알고리즘에 대한 연구와 WBMP와 같은 새로운 형식의 컨텐츠를 효과적으로 압축 할 수 있는 압축 알고리즘의 개발에 대한 연구가 필요하다.

참 고 문 헌

[1] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. Hypertext Transfer Protocol-HTTP/1.1, Internet Draft draft-ietf-http-v1.1-spec-07, August 1996.

[2] WAP Forum, Wireless Application Protocol Architecture, April 1998, http://www.wapforum.org/.

[3] WAP Forum, Wireless Session Protocol, April 1998, http://www.wapforum.org/.

[4] 최은정, 한동원, 임경식, "무선 인터넷 서비스를 위한 WAP 게이트웨이용 WML 컴파일러의 설계 및 구현", 한국정보과학회 논문지 : 컴퓨팅의 실재, 제 7권, 제 2호, pp.165-182, 2001년 4월.

[5] Eetu Ojanen, Jari Veijalainen, "Compressibility of WML and WMLScript byte code: Intitial results," Proceedings of the 10th International Workshop on Research Issues in Data Engineering, February 2000.

[6] WAP Forum, Wireless Transaction Protocol, April 1998, URL:http://www.wapforum.org/.

[7] L. Svobodova, "Implementing OSI systems," IEEE Journal on Selected Areas in Communications, vol. 7, No 7, pp.1115-1130, September 1989.

[8] Douglas C. Schidt, "Transport System Architecture Services for High Performance Communications Systems," IEEE Journal on Selected Areas in Communications, vol. 11, no. 4, pp. 489-506, May 1993.

[9] 오행석, 이부호, 손홍세, 최영한, 박용범, 이준원,김성운, 정보통신 프로토콜 공학, 한국전자통신연구원, January 1998.

[10] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, alachander Krishnamurthy, "Potential Benefits of Data Encoding and Data Compression for HTTP," Procedings of SIGCOMM'97, pp.181-194, September 1997.

[11] P. Deutch, GZIP File Formt Specification Version 4.3, RFC 1952, Aladdin Enterprises, May 1996.

[12] Wei Yu, Ruibiao Qiu, Jason Fritts, "Evaluation of Motion-JPEG2000 for Video Processing," Technical Report of Department of Computer Science in Washington University, http://www.cs.wustl.edu/cs/techreports/2001/wucs-01-34.ps.gz, November 2001.

[13] Maxime flament, Arne sevensson, "Interference Mitigation in 60 GHz Wireless Local Area Networks," Proceedings IEEE Vehicular Technology Conference Fall, Vancouver, Canada, September 2002 submitted.

[14] WAP Forum, WAP 2.0 Architecture, October 2000, URL:http://www.wapforum.org/.

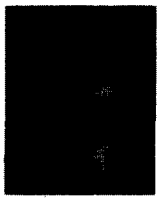
[15] David Clark, "Perparing for a New Generation of Wireless Data," Computer, Vol. 32, No. 8, pp.8-11, August 1999.

[16] Neal Leavitt, "Will WAP Deliver the Wireless Internet?," Computer, Vol. 33, No. 5, pp.16-20, May 2000.

[17] David J. Goodman, "The Wireless Internet: Promises and Challenges," Computer, Vol. 33, No. 7, pp.36-41, July 2000.

김 기 조

1999년 경북대학교 컴퓨터과학과(이학사). 2002년 경북대학교 컴퓨터과학과(이학석사). 2002년 ~ 현재 경북대학교 컴퓨터과학과 박사과정. 관심분야는 이동 컴퓨팅, 무선 세션 프로토콜, HTTP, 윈도우즈 CE 커널, 디바이스 프로그래밍



이 동 근

2001년 경북대학교 컴퓨터과학과(이학사). 2001년 ~ 현재 경북대학교 컴퓨터과학과(석사과정). 관심분야는 무선 인터넷, 네트워크 보안, 컴퓨터통신

임 경 식

정보과학회논문지 : 정보통신
제 29 권 제 4 호 참조