유선망에서의 UDP/IP 헤더압축 프로토콜의 구현 및 성능분석

나종민* · 이종범** · 이인성*** · 신병철***

Implementation and Performance Analysis of UDP/IP Header Compression Protocol in Wired Networks

Jong-min Na* · Jong-Bum Lee** · In-sung Lee*** · Byung-Cheol Shin***

이 논문은 2003년도 한국과학재단 목적기초연구과제(R01-2003-000-11620-0)지원으로 수행되었습니다.

요 약

현재의 인터넷 환경은 실시간 서비스와 멀티미디어 데이터의 처리 요구들이 계속 늘어나고 있는 추세이다. 그런데, 현재 널리 쓰이는 UDP/IP 프로토콜의 헤더 부분에는 상당한 오버헤드가 존재하고 있다. 즉, 같은 패킷 스트림 안에서 연속적인 패킷의 헤더 사이에 중복된 부분이 많다. 헤더 압축은 바로 이러한 오버헤드를 최소화하여 전송 효율을 높이는 방법이다. 거의 변화하지 않는 필드 정보를 처음에 한번 보낸 후 송·수신단 사이에 계속 유지함으로써 그 이후의 패킷에서 요구되는 헤더의 일부분을 미리 예상 할 수 있다. 이렇게 함으로써 전송이 요구되는 UDP/IP 헤더의 크기를 최소화 할 수 있다.

본 논문에서는 유선, 이더넷 환경에서 UDP/IP 프로토콜의 헤더를 압축하였다. 검토결과 대부분의 UDP/IP 헤더는 13 바이트를 줄인 7 바이트 정도로 압축될 수 있었으며, 제안된 헤더 압축 시스템은 리눅스 환경에서 고안되고 구현되었다. Ethernet 환경에서는 payload 길이가 최소한 64 바이트 이사이어야 하므로 실제 채팅 환경에서는 6.6 바이트를 헤더에서 줄일 수 있었다.

ABSTRACT

Recently, the demands for real-time service and multimedia data are rapidly increasing. There are significant redundancies between header fields both within the same packet header and in consecutive packets belonging to the same packet stream. And there are many overheads in using the current UDP/IP protocol. Header compression is considered to enhance the transmission efficiency for the payload of small size. By sending the static field information only once initially and by utilizing dependencies and predictability for other fields, the header size can be significantly reduced for most packets. This work describes an implementation for header compression of the headers of IP/UDP protocols to reduce the overhead on Ethernet network. Typical UDP/IP Header packets can be compressed down to 7 bytes and the header compression system is designed and implemented in Linux environment. Using the Header compression system designed between a server and clients provides have the advantage of effective data throughput in network. Since the minimum packet size in Ethernet is 64 bytes, the amount of reduction by header compression in practical chatting environment was 6.6 bytes.

키워드

ROHC, UDP, IP, 헤더압축

*인소팩(주)

**(주)코다컴테크놀러지

***충북대학교

접수일자: 2004. 06. 13

Ⅰ. 서 론

최근 과학 기술의 발전과 더불어 컴퓨터를 이용한 통신 기술의 발전으로 광섬유를 전송 매체로 하는 고속통신망의 등장과 함께 트래픽의 크기도 예전의 텍스트 중심이었던 것과 다르게 대용량의 멀티미디어 트래픽이 증가하고 있다. 현재 많이 사용되고 있는 컴퓨터 데이터용 TCP/IP나 실시간 서비스에 널리 사용되는 RTP/UDP/IP프로토콜은 고속화된 망 하부 구조의 속도를 제대로 이용하지 못하는 실정이다. 이 프로토콜들이 설계될 당시에는 에러가 많고 대역폭이 적은 망 환경특성에 맞게 설계되어 에러 제어가 주된 관심사였으나, 에러가 극히 적고 대역폭이 매우 큰 고속망 환경에서는 복잡한 에러 제어 방법 때문에 망이 제공하는 빠른 속도를 제대로 활용하지 못하고 있는 실정이다[1][2]. 이러한 제한된 대역폭에서의 문제점을 해결하기 위한 기술들은 유선망보다는 대역폭이 좁은 무선망에서 활발히 연구되고 있다. 대표적인 기술은 Header Removal/ Generation, Header Striping/ ReGeneration, 헤더 압축기술[3] 등이 있으며, 여러 가지 기술 중에서 가장 이슈가 되는 기술이 헤더 압축 기술이다. 이 헤더 압축 연구는 PPP (Point-to-Point Protocol)프로토콜 기반 환경에서의 연구가 중심이 되고 있다. 유선망환경에서는 TCP/IP프로토콜의 헤더 압축[4][5], RTP/UDP/IP 프로토콜의 헤더 압축[6]이 연구되었으며, 무선망에서는 RTP/ UDP/IP 프로토콜의 헤더를 압축하는 ROHC헤더 압축[7]을 중심으로 연구되고 있다. [8]에서는 헤더압축 및 이동호스트의 핸드오프 발생시 신속하게 압축상태를 복구하는 기법에 대한 연구를 수행하였다. TCP/IP 프로토콜이나 실시간 통신에서 사용되는 RTP/UDP /IP 프로토콜에서 대부분의 헤더 필드 값들은 다음과 같은 몇가지 특징을 갖는 것으로 구분된다. 일정한 필드 값, 증가폭이 일정한 필드 값, 다른 필드 값을 통해 추론이 가능한 필드 값 등으로 구분된다. 이러한 특성을 이용하여 각 프로토콜의 오버헤드를 줄이게 된다. 헤더 압축 기술을 이용하여 이더넷 기반의 유선 망 환경에 알맞게 적용을 하면, 전송되는 오버헤드들을 감소시켜서, 네트워크의 부하를 줄여서 효율을 증가시킬 수 있다. 특히 인터넷 채팅에서와 같이 전송되는 데이터의 크기가 작은 실시간 서비스에서는 헤더의 크기가 데이터 크기에 비하여 상대적으로 크다고 할 수 있는데, 이런 상황에서 헤더 압축은 더욱 효율적이다.

본 논문에서는 UDP/IP 헤더를 이더넷 기반의 유선망 환경에 맞도록 헤더 필드들을 클래스별로 분류하고 헤더 압축을 설계하여 리눅스 플랫폼에서 구현하였다. 논문의 구성은 다음과 같다. 2장에서는 현재 사용하고 있는 UDP/IP 프로토콜의 문제점 및 해결 방안으로써 헤더 압축의 기본 개념과 최적화시키기 위한 헤더 필드 분류와 기존의 헤더 압축 기술의 개념에 대해서 설명하고, 3장에서는 헤더 압축 구현의 기반이 되는 리눅스 커널의 전송 및 네트워크 계층의 송수신 부분을 분석하고, 송수신되는 패킷을 관리하는 소켓 버퍼의 구조에 대해서 살펴본다. 4장에서는 헤더 압축 설계와 구현에 대해서 설명하며, 5장에서는 헤더 압축에 따른 성능 평가를 하기 위하여, 기존의 UDP/IP프로토콜을 이용한 채팅 서비스와 헤더 압축을 이용한 채팅 서비스를 일정 시간동안 서비스한 후에 채팅에 사용된 데이터를 수집하여 실험 결과를 평가하고, 6장에서는 실험 결과에 대하여 고찰하며 결론을 맺는다.

Ⅱ. 헤더 압축의 기본개념

(1) 기존 UDP/IP 프로토콜의 문제점

현재 사용되고 있는 UDP/IP/이더넷 프로토콜을 이용하며, 서비스를 하는 경우, 헤더의 크기는 UDP 8바이트, IP 20바이트, 이더넷 14바이트이다. 그림 1을 보면 이더넷의 최소 전송 크기인 46바이트[9]에서 UDP/IP헤더의 크기를 제외하면, 18바이트가 남으며, 전송하는 데이터가 1~17바이트까지는 패딩이 붙게 된다. 데이터가 42바이트 이하이면, 데이터 보다 큰 오버헤드가 붙는다는 것을 알 수 있다. 이더넷의 최소 전송

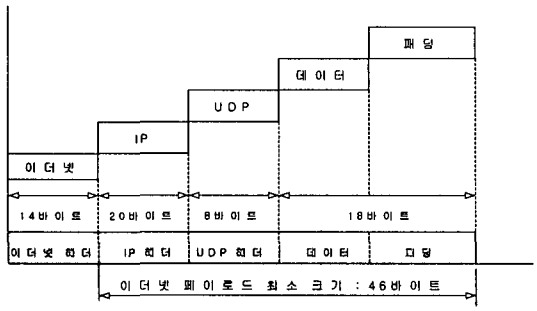


그림 1 이더넷 최소 전송 크기
Fig.1 Minimum packet size of Ethernet

데이터의 46바이트인 경우 UDP/IP/이더넷의 42바이트의 헤더와 18바이트의 데이터로 데이터의

크기의 2배 이상의 오버헤드가 전송되게 된다.

(2) 헤더 필드의 분류

보통 필드들을 크게 INFERRED, STATIC, STATIC_KNOWN, STATIC_DEF, CHANG- ING으로 분류하고 있으며[10], 그 의미는 다음과 같다.

- -INFERRED : 길이 필드처럼 다른 헤더필드들을 통하여 추론이 가능한 필드.
- -STATIC : 패킷 스트림의 수명동안 일정한 값을 가지는 필드.
- -STATIC_KNOWN : RTP나 IP의 버전 필드처럼 잘 알려진 값들을 가지는 필드.
- -STATIC-DEF : IP 주소나 포트 번호 등 헤더 필드에 STATIC하게 정의되는 필드.
- -CHANGING : 필드들은 랜덤한 값을 가지는 필드들을 말하며 변화되는 값에 따라 STATIC, SEMISTATIC, RARELY-CHANGING, ALTERNATING, IRREGULAR등 다섯 개의 Sub-class를 가지고 있다.

표 1은 ROHC[7]에서 IP헤더 필드를 분류한 것이다.

표 1 IP 헤더 필드의 분류
Table 1 Classification of header fields

Filed	size(bits)	Class
Version	4	STATIC-KNOWN
Header Length	4	STATIC-KNOWN
ToS	8	CHANGING
Total len	16	INFERRED
IP-ID	16	CHANGING
Reserved flag	1	STATIC-KNOWN
May Fragment flag	1	STATIC
Last Fragment	1	STATIC-KNOWN
Fragment offset	13	STATIC-KNOWN
TTL	8	CHANGING
Protocol	8	STATIC-KNOWN
Header Checksum	16	INFERRED
Source Address	32	STATIC-DEF
Destination Address	32	STATIC-DEF

헤더 필드를 분류하는데 있어서 효율적인 헤더 압축을 위하여, 표 2처럼 통계적으로 나타난 데이터를 이용하여 헤더 필드들을 분류[10]하기도 한다.

표 2 헤더 필드들의 통계적인 확률 수치
Table 2 Classification probability of header fields

Field	Class	Probability
Version Header length Reserved flag Last fragment Fragment offset Source address Destination address	STATIC	100%
Header checksum Protocol Total len	INFERRED	100%
TTL	STATIC	99%
	CHANGING	1%
ToS	STATIC	99.9%
May fragment flag	CHANGING	0.1%
ID	INFERRED	99%
	CHANGING	1%

III. 유선망에서의 헤더 압축 설계 및 구현

(1) 필드 분석 및 분류

헤더 필드의 분류는 대부분 무선망에서의 헤더 필드 분류와 비슷하며, 차이점은 무선망에서 대부분 디바이스 계층으로 PPP를 사용하기 때문에 라우팅이 필요없지만, 유선망의 환경에서 이더넷을 사용할 경우, 라우터를 통과해야하기 때문에 라우터가 헤더압축을 지원하지 않는 경우에는 라우팅에 관련된 정보를 모두 전송하여야 한다.

실제 ftp서비스나 채팅 서비스등 TCP/IP프로토콜과 UDP/IP프로토콜의 패킷들을 캡쳐하여 분석한 결과 대부분은 무선망에서 정의한 필드들과 같았지만 몇몇 필드의 경우에는 기존 헤더 분류의 경우와 다른 경우가 발생하였다. IP프로토콜의 경우에 헤더 패킷을 분석해 보면 표 3과 같다. 대부분 필드에 할당된 클래스가 무선망에서의 할당된 필드 클래스와 비슷하지만 ToS는 리눅스나 윈도우 기반의 시스템에서는 지원이 안 되고 있으며, IP-ID필드의 경우에는 TCP는 1씩 증가 하지만, UDP의 경우에는 0값을 갖는다. 그러므로 추론이 가능하거나 STATIC값을 갖는다. Frag_off의 경우

에도 STATIC값을 가지고 있다.

무선망 환경에서는 에러율이 높은 관계로 헤더 체크섬이 필요하지만, 유선망에서는 무선망의 환경과는 달리 에러율이 낮다. 표 4에서 보는 바와 같이, 유선망에서 각 프로토콜의 에러율이 매우 낮다[11]. 따라서 만약에 체크섬이 필요하다면 현재 사용하고 있는 체크섬 16비트 보다 적은 비트로 설정하여 오버헤드를 줄이는 것이 좋을 것이다. 무선망에서도 전송 에러가 낮은 경우에는 체크섬 없이 전송하며, 전송 에러가 높은 경우에는 3~4비트 정도의 체크섬[12]을 전송하고 있다.

표 5에서 보는 바와같이, UDP 헤더 필드에 대해서 살펴보면, 응용 계층에서 서비스가 시작해서 끝나기까지 포트 번호는 일정하다. UDP 길이 필드는 추론이 가능한데, 만약 IP의 전체 길이 값을 알고 있다면, TCP와 달리 대부분 IP option이 전송되지 않기 때문에 전체 길이에서 IP헤더 길이를 제외하면 UDP 길이 필드가 되며, 여기에 UDP헤더 길이를 제외하면, 순수 데이터만 남게 된다. UDP의 체크섬은 옵션으로 설정되어 있기 때문에 에러율이 낮은 경우 생략할 수 있다.

표 3 실제 시스템에서 IP 헤더 패킷 분석
Table 3 Classification of IP packet header

Filed	size(bits)	Class
Version	4	STATIC
Header length	4	STATIC
ToS	8	STATIC
Total len	16	INFERRED
IP-ID	16	STATIC
frag_offset	16	STATIC
TTL	8	STATIC
Protocol	8	STATIC
Header checksum	16	INFERRED
Source address	32	STATIC
Destination address	32	STATIC

표 4 유선망에서 각 프로토콜에서의 에러율
Table 4 Error rate of each protocol in wired networks

프로토콜	전송된 패킷	CRC 에러	에러율(%)
Ethernet	170,000,000	446	2.62×10^{-5}
IP	170,000,000	14	8.23×10^{-8}
UDP	170,000,000	6	3.57×10^{-8}
TCP	170,000,000	1983	1.16×10^{-5}

표 5 실제 시스템에서 UDP 헤더 패킷 분석
Table 5 Classification of UDP header packet

Field	size(bits)	Class
Source port	16	STATIC
Destination port	16	STATIC
Length	16	INFERRED
Checksum	16	INFERRED

헤더 압축 설계는 기존의 IP/UDP헤더 필드들을 이용하여 헤더 압축을 하였으며, IPv4에 대해서만 고려하여 설계하였다. ip_forwarding을 하는 라우팅 시스템에서 헤더 압축을 지원하는 경우와 지원하지 않는 경우의 두 가지 모델로 나누었으며, 각 모델에 따라 위에서 정의한 필드들의 분석을 토대로 전송이 필요한 필드들은 다음과 같다:

- Model I : ip_forwarding을 헤더압축이 지원하는 경우

서비스 구별 : CID(Context Identifier, 1바이트) + 발신지 IP 주소.

UDP 길이 필드와 소켓 버퍼는 예상가능하고 나머지 UDP/IP 필드는 제거.

그림 2는 Model I과 같이 ip_forwarding을 헤더 압축이 지원하는 경우에 제거 가능한 UDP/IP헤더 필드들을 보여 준다. UDP헤더 필드 8바이트와 IP 헤더 13바이트를 제거할 수 있으며, 제거되는 헤더 필드 대신에 데이터가 전송되어 전송효율을 높이게 된다.

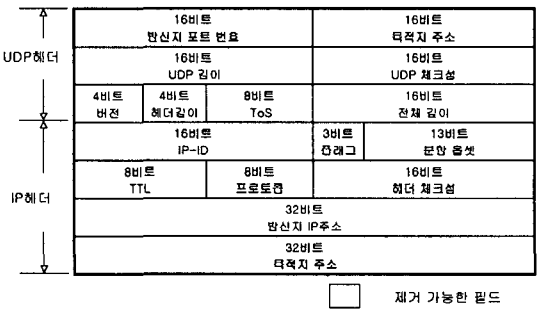


그림 2 Model I의 경우 제거 가능한 헤더 필드
Fig.2 Erasable header fileds in Model I

- Model II : ip_forwarding을 헤더압축이 지원하지 않는 경우

ip_rcv에서 필요한 필드 : IP 버전+IP 헤더 길이 +IP 체크섬.

라우팅에 필요한 필드 : 발신지/목적지 IP 주소, ToS, TTL. ToS 필드는 일반적으로 사용되지 않지만 특별한 경우 라우터에서 사용될 수도 있기 때문에 전송한다.
서비스 구별 : CID(1바이트). CID는 Stream을 구분할 수 있도록 하기위하여 전송됨.

UDP 길이 필드와 소켓 버퍼는 예상할 수 있는 값이며, 나머지 UDP/IP 필드는 제거한다.

그림 3은 ip_forwarding을 헤더압축이 지원하는 않는 경우(Model II)에 제거 가능한 UDP/IP헤더 필드들을 보여 준다. 그림에서 보는바와 같이 UDP헤더에서 8바이트와 IP헤더에서 4바이트를 제거할 수 있다. 제거되는 필드에는 대신 데이터가 전송된다.

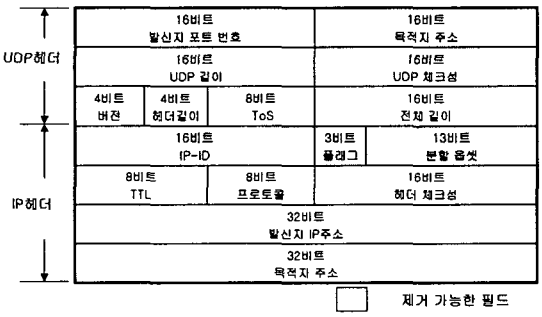


그림 3 Model II의 경우 제거 가능한 헤더 필드
Fig. 3 Erasable header fields in Model II

(2) 헤더 압축 설계 및 구현

본 논문에서 헤더 압축의 성능을 평가하기 위하여 와우 리눅스 7.1(커널 2.4.2)기반의 환경에서 ip_forwarding의 헤더압축 지원 여부에 따라 두 가지의 모델로 나누어 코드를 구현하고 성능을 평가하였다. 구현 환경은 표 6에서 같이 하나의 서버와 두 종류의 클라이언트 모델로 Model I의 경우, 클라이언트와 서버 중간에 라우터 없이 전송되며, Model II의 경우, 클라이언트와 서버 중간에 IP주소가 210.125.158.1인 라우터를 거치게 된다.

표 6 헤더 압축 구현 환경
Table 6 Implementaion environment of header compression

name	Server	Client1	Client2
OS	WOW linux 7.1	WOWlinux 7.1	WOWlinux 7.1
Kernel	2.4.2	2.4.2	2.4.2
IP 주소	210.125.151.114	210.125.151.121	210.125.147.124
용도	Server	Client	Client
위치	실험실	실험실	실험실

구현 범위는 IPv4에만 한정을 하였고, 에러율을 고려하지 않고 구현 하였다. 디바이스로는 Ethernet을 사용하였기 때문에, 최소 페이로드 단위는 46바이트[4]로 IP헤더 20바이트, UDP헤더 8바이트가 된다. 그러므로 순수 데이터가 18바이트(46바이트-20바이트-8바이트) 이하일 때 헤더 압축은 의미가 없게 된다. 따라서 데이터가 19바이트 이상인 경우에 한하여 헤더 압축을 하게 된다.

전송 계층의 프로토콜에 따라서 IP계층에서 IP 헤더를 생성하는 함수와 헤더 생성 후 경유하는 함수는 서로 다르다. 서로 다른 방법으로 헤더를 생성한 후에 모두 IP 계층의 ip_output함수를 통과하여 하위 계층으로 전송된다. 그림 4에 설계된 헤더 압축 시스템은 IP 계층의 ip_build_xmit에서 IP헤더의 필드들을 생성하고, 소켓 버퍼에 전송 후, 디바이스 계층으로 내려가기 전에 ip_header_comp에서 소켓 버퍼에 있는 데이터와 UDP/IP헤더를 복사 후에 데이터를 압축한 후에 다시 소켓 버퍼로 복사하고, 이더넷을 통해 전송을 하게 된다. 패킷을 받은 수신단의 ip_rcv에서도 압축된 헤더를 소켓 버퍼에 복사한 후 원래의 헤더로 복구한 후에 다시 소켓 버퍼에 복사한 후 ip_rcv_finish함수로 전달하게 된다. 목적지 IP 주소에 따라 자신의 주소인 경우 UDP 계층으로 보내고, 목적지가 자신이 아닌 경우 ip_input_route함수와 ip_output함수를 통해 목적지로 전송하게 된다. ip_forwarding이 헤더압축을 지원하지 않는 경우에는 압축된 헤더의 decompression과정이 없이 라우팅 테이블을 통해 ip_forwarding 하게 된다.

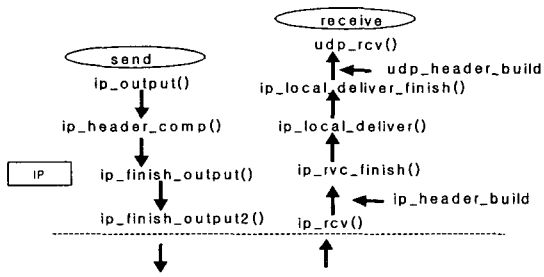


그림 4 설계된 헤더 압축의 흐름
Fig. 4 Designed Flow for header compression

헤더 압축의 전체적인 흐름을 보면 그림 5와 같이 IP의 전체길이를 확인하고, 전체 길이가 46이하이면, 헤더 압축 없이 ip_finish_output함수를 호출하여 전송하게 되고, 46을 초과하면 헤더 압축을 하게 된다. 소켓 버퍼를 수정해야 되기 때문에 먼저 원본의 소켓 버퍼를 소켓버퍼2로 복사하고, 소켓 버퍼를 초기화하여 내부의 헤더 및 데이터를 제거한다. 데이터를 제거한 후 정의된 헤더 필드에

맞게 소켓 버퍼를 생성한다. 생성 후 ip_finish_output 함수로 전달하게 된다.

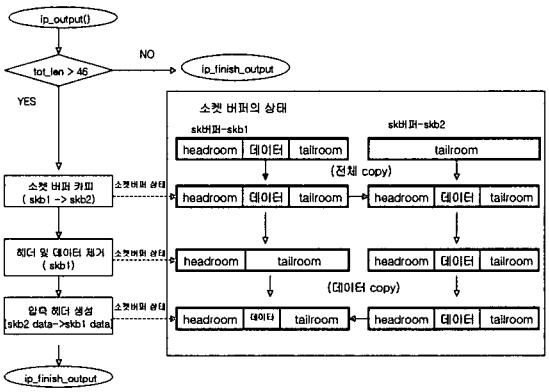
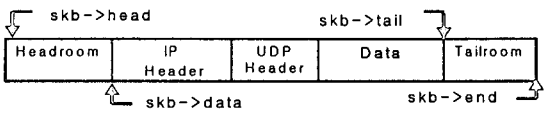


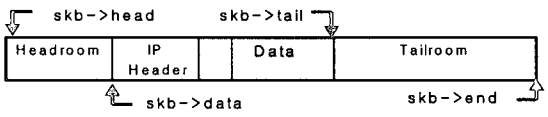
그림 5 구현된 헤더 압축의 블록도
Fig. 5 Block diagram of implemented header compression

지금부터는 소켓 버퍼의 저장구조를 살펴보기로 한다. 첫 번째로, ip_forwarding을 헤더압축이 지원하는 경우(Model I), IP 계층에서 압축전의 소켓 버퍼에 저장된 구조를 살펴보면 그림 6의 (a)와 같다. 앞의 headroom은 디바이스 헤더를 위하여 미리 할당된 공간이며, 그 다음 IP헤더, UDP헤더, 응용 계층에서 내려온 데이터가 저장되어있고, 아직 할당되지 않은 tailroom이 남아있게 된다.

저장된 소켓 버퍼를 헤더 압축 후에는 그림 5와 같이 Headroom의 경우에는 변함이 없으며, IP헤더의 필드들은 위에 정의된 것과 같이 IP헤더 일부 대신 데이터가 들어가고, UDP헤더 또한 제거되어 데이터가 들어가게 되어 Tailroom의 크기가 커지게 된다.



(a) IP 계층에서의 소켓 버퍼의 본래 모습



(b) IP 계층에서 압축된 소켓 버퍼
그림 6 IP 계층에서의 본래의 소켓버퍼와 압축된 소켓버퍼
Fig. 6 Original and compressed socket buffers in IP layer.

디바이스 계층에서는 그림 6의 (b)과 같은 소켓 버퍼를 받아서 전송하게 된다. 그 이후의 흐름은 앞의 헤더 압축 시스템 흐름에서 설명한 것과 같이 ip_forwarding이 되는 서버에서는 ip_rcv 함수에서 ip 헤더를 생성하고 IP주소에 따라 라우팅 테이블을 이용하여 라우팅 정보를 얻고, 다시 헤더 압축 후에 ip_output을 거쳐서 다시 목적지로 전송하게 된다. 최종 수신 측에서도 마찬가지로 ip_rcv에서 압축 헤더를 원래의 헤더로 복원하고, 상위 계층인 udp_rcv함수로 보내서, UDP 헤더를 다시 만들게 된다.

두 번째로 ip_forwarding을 헤더압축이 지원하지 않는 경우(Model II), 압축된 헤더를 생성하는 절차는 똑같고, 다른 점은 ip_forwarding을 하기 위해서는 많은 제약이 따르게 된다. 라우팅 되는 시스템에서 drop되지 않고, 라우팅 테이블을 이용하려면, 많은 정보를 전송해주어야 된다. 먼저 ip_rcv에서 drop되지 않기 위해서는 3가지의 조건을 확인하게 된다. IP 버전, IP 헤더 길이를 확인하고, 전체길이와 버퍼의 길이를 비교하여 전체 길이보다 버퍼의 길이가 작은 경우, 패킷의 일부가 손상 되었을 거라고 판단하고 drop하게 된다. 마지막으로 IP 체크섬을 계산하여, 에러가 없는지 확인하게 된다. 위의 3가지 조건을 모두 만족시켜주도록 하기 위해서는 IP 버전과 IP 헤더 길이는 전송해주어야 하며, 전체 크기를 헤더 압축후의 소켓 버퍼보다 크지 않도록 재조정을 해줘야 하며, IP 헤더 필드 값들을 수정후 IP 체크섬을 다시 생성해야한다. 그리고 라우팅 테이블을 이용하기 위하여 필요한 필드들은 IP 발신지/목적지 주소, TTL, ToS이며, 위와 같은 필드들은 전송되어야만 한다.

IV. 적용 및 분석

본 논문에서 구현된 헤더 압축에 따른 성능 평가를 하기 위하여, 일정 시간동안 기존의 UDP/IP 프로토콜을 이용한 채팅 서비스와 헤더 압축을 이용한 채팅 서비스를 실시한 후에 채팅에 사용된 데이터를 수집하여 데이터에 대한 오버헤드비율로 성능을 측정하였다.

첫 번째로 Model I의 경우(서버와 클라이언트가 라우터를 거치지 않는 같은 세그먼트 안에 있는 경우)를 살펴본다. 그림 7의 Model I 그래프를 보면 기존 UDP/IP프로토콜 헤더의 경우 19바이트부터 데이터 크기에 따라 패킷 크기가 증가하지만, 헤더 압축을 한 경우에는 40바이트부터 패킷 크기가 증

가하게 된다. 40바이트부터 증가하게 되는 이유는 헤더 압축을 통해 UDP/IP헤더 21바이트를 감소시킬 수 있기 때문이다.

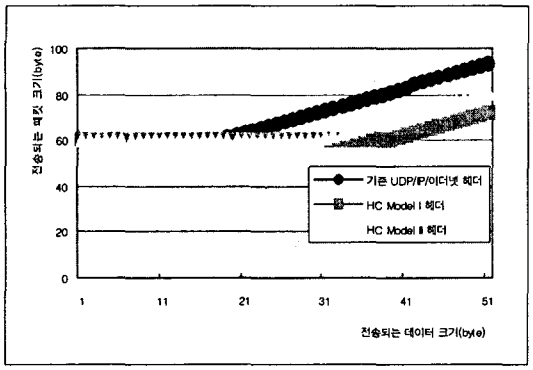


그림 7 Non-HC와 HC경우의 전송 헤더 크기 비교
Fig.7 Comparison of header size between Non-HC and HC

성능을 평가하기 위하여 UDP 채팅 서버를 5분간 서비스를 하여 1분 단위로 전송되는 패킷 데이터를 수집 분석하였다. 표 7을 보면 전송된 패킷수는 분당 전송되는 패킷의 수를 말하며, IP 전체 길이 합은 전송된 데이터와 전송된 헤더의 합을 말한다. 전송된 데이터는 패킷 중 데이터 부분의 크기를 말하며, 전송된 헤더는 UDP/IP 프로토콜의 헤더 크기의 합을 말한다. 전송 효율을 평가하기 위하여 도입된 오버헤드율(%)은 전송된 데이터에 대한 오버헤드 비율을 말하며, 데이터의 길이가 이더넷의 최소 페이로드보다 작아 붙는 패딩도 데이터로 취급하였다. 오버헤드 비율이 낮을수록 데이터 전송 효율이 높아진다.

그림 8은 표 7의 시간변화에 따른 오버헤드율을 그래프로 나타낸 것으로 그래프를 보면 헤더 압축을 하게 되면 시간마다 조금씩 차이는 있지만 약 50%정도의 오버헤드를 줄일 수 있게 되며, 패킷당 평균 10바이트 정도의 헤더를 감소시킨다. 최대 패킷 감소 값인 21바이트와는 많은 차이를 나타내는데, 이유는 전송되는 패킷의 데이터 크기에 따라 헤더 압축 효율이 많이 달라지기 때문이다. 최대 헤더압축(21바이트의 헤더 감소) 경우와 비슷한 효율을 가지려면 전송되는 패킷의 IP 전체 길이 값이 67바이트 이상의 값을 가져야 되지만 실제 실험에서는 67바이트 보다 작은 데이터의 전송의 빈도에 따라 효율이 달라진다.

표 7 Model I 환경에서 채팅 서버의 전송 패킷 분석
Table 7 Packet analysis of chatting server in Model environment

	1분	2분	3분	4분	5분	평균
전송된 패킷수	24	22	21	25	21	23
IP 전체 길이 합 (byte)	1200	1080	1001	1212	1023	1103
전송된 데이터(byte)	528	464	413	512	435	470
전송된 헤더(byte)	672	616	588	700	588	633
전송된 HC 헤더 (byte)	404	391	399	456	378	406
기존 UDP/IP오버헤드율(%)	127	133	147	137	135	135
HC의 경우, UDP/IP 오버헤드율(%)	77	84	97	89	87	87

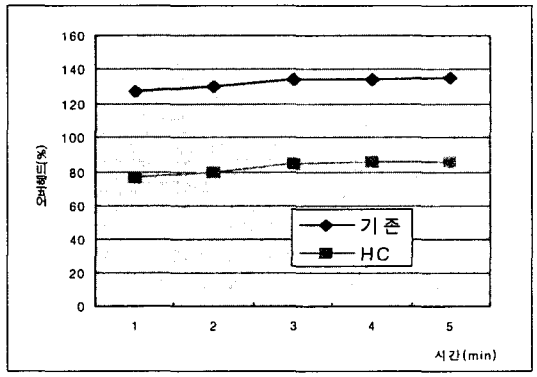


그림 8 Model I 환경에서 채팅 서버의 오버헤드율(%)
Fig. 8 Overhead rate of chatting server in Model I environment

두 번째로 Model II의 경우(서버와 클라이언트가 라우터를 거치는 다른 세그먼트에 있는 경우)를 살펴본다. 그림 7의 Model II 그래프를 보면 전송되는 데이터가 18바이트가 되기 전 까지는 패딩을 포함시켜 이더넷의 최소 페이로드인 46바이트가 전송된다. 일반적인 UDP/IP헤더의 경우 19바이트부터 데이터의 크기에 비례하여 증가하지만, 헤더 압축을 한 경우에는 31바이트부터 데이터의 크기에 비례하여 전송되는 패킷의 크기가 증가하게 된다.

위의 첫 번째 실험과 동일한 방법으로 UDP 채팅 서비스를 5분간 실시하여 1분 단위로 전송되는 패킷의 데이터를 분석하였다. 표 8의 결과를 보면 기존의 UDP/IP프로토콜의 오버헤드율이 평균 161%인데 반하여, 헤더 압축을 한 경우 123%의 오

버헤드율이 발생하여 약 40%의 오버헤드를 감소시켰다.

그림 9는 표 8의 시간에 대한 오버헤드율을 그래프로 나타낸 것으로 헤더 압축을 하게 될 경우 약 40%의 오버헤드율이 감소되며, 이를 바이트로 환산하면 패킷 당 평균 6.6바이트 정도의 헤더를 감소시킨다. 이 실험도 역시 최대패킷 감소 값인 12바이트 값과 차이를 보이는데, 위의 실험에서와 마찬가지로 전송되는 패킷의 IP 전체 길이 값이 58바이트보다 작은 데이터 전송이 많기 때문이다.

그림 10은 앞 실험의 결과를 데이터 크기에 따른 오버헤드율로 도식화한 그림이다.

표 8 Model II 환경에서 채팅 서버의 전송 패킷 분석
Table 8 Packet analysis of chatting server in Model II environment

	1분	2분	3분	4분	5분	평균
전송된 패킷수	24	26	25	27	23	25
IP 전체 길이 합(byte)	1128	1129	1168	1189	1071	1137
전송된 데이터(byte)	456	401	468	433	427	437
전송된 헤더(byte)	672	728	700	756	644	700
전송된 HC 헤더(byte)	497	571	517	599	484	534
기존 UDP/IP 오버헤드율(%)	147	182	150	175	151	161
HC UDP/IP 오버헤드율(%)	109	142	110	138	113	123

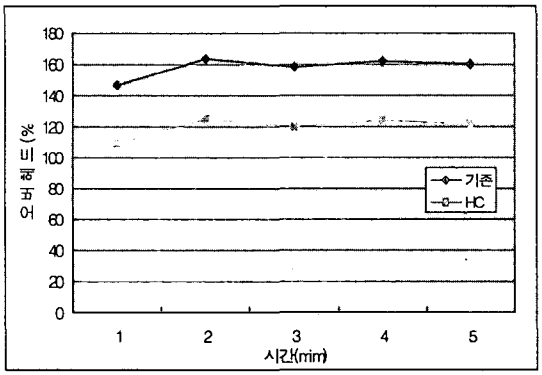


그림 9 Model II 환경에서 채팅 서버의 오버헤드율(%)
Fig. 9 Overhead rate of chatting server in Model II environment

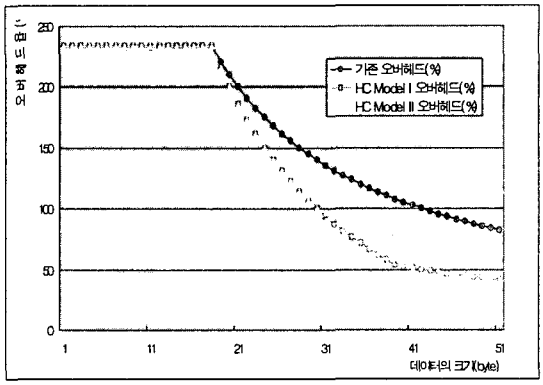


그림 10 Non-HC와 HC경우의 오버헤드율(%) 비교
Fig. 10 Comparison of overhead rate between Non-HC and HC

V. 결 론

실시간 서비스에서 데이터의 크기가 작은 경우에는 추가되는 헤더가 일정하기 때문에 상대적으로 오버헤드가 커지게 된다. 이런 이유에서 헤더 압축을 하게 되는데 이러한 분야의 연구는 유선망에 비하여 대역폭이 상대적으로 비싼 무선망에서 활발히 수행되고 있다. 본 헤더 압축에 대한 연구에서는 무선망에 이용되는 헤더 압축을 이더넷기반의 유선망에 맞게 헤더 필드들을 정의한 후, 두 가지 환경에 따라서 헤더 압축을 한 결과 헤더의 크기를 10~20byte정도 감소시킬 수 있으며, 오버헤드를 데이터의 40~50%정도 감소시키는 결과를 보였다. 이러한 헤더 압축 기술은 작은 데이터가 빈번하게 발생하는 경우에 사용하게 된다면 매우 효율적인 기술이 될 것이다. 유선망과 이더넷이라는 환경 적인 요소로 인하여 무선망에서의 연구와는 효율적인 면에서 차이가 있지만 무선망에서와 같이 높은 효율을 얻을수 있도록 좀더 연구를 진행해야 할 것이다. 지금 까지 구현한 헤더 압축 시스템을 바탕으로 헤더 압축 범위를 RTP프로토콜까지 확장한다면 무선망에서의 연구와 비슷한 효율을 얻을 수 있을 것이라 생각한다. 차후 연구 방향으로는 IP sec을 고려하여 압축하는 것과 헤더 압축 범위를 RTP프로토콜로 확장을 하여 RTP/UDP/IP를 실시간 서비스에 동작하도록 구현하는 것이다. RTP/UDP/IP 헤더 압축과 보안을 통한 실시간 음성 서비스를 하게 된다면 다른 실시간 서비스관련 응용 프로그램의 개발하는데 있어서 기초 및 연구

자료가 될 수 있다.

참고문헌

[1] W. Doeringer, D. Dykeman, M. Kaiserwerth, B. Meister, H. Rudin, and R. Willianmson, "A survey of light-weight transport protocols for high-speed network," IEEE Trans. Commun., vol.38, no.11, pp.2025-2039, Nov. 1990.

[2] T. Porta and M. Schwartz, "Architecture, features, and implementation of high-speed transport protocols," IEEE Network Mag., pp.14-22, May 1991.

[3] TSG-SA-WG2, TSG-GERAN and TSG-RAN, "Header compression for optimized voice bearers," Joint Meeting, 3GPP, Aug. 2001.

[4] V. Jacobson, "Compressing TCP/IP headers for low-speed serial link," IETF RFC 1144, Feb. 1990.

[5] M. Degermark, B. Nordgern and S. Pink, "IP header compression," IETF RFC 2507, Feb. 1990.

[6] S. Casner and V. Jacobson, "IP/UDP/RTP headers for low_speed serial links," IETF RFC 2508, Feb. 1999.

[7] C. Borman et. al., "Robust Header Compression(ROHC)," IETF RFC 3095, July 2001.

[8] 강문식, "이동성 IP 기반 네트워크에서 헤더 압축기법을 이용한 효율적인 비디오 트래픽 관리기법," 한국통신학회논문집 vol.26 no. 9A, pp.1583-1591, 2001.

[9] C. Hornig, "A standard for the IEEE transmission of IP datagrams over Ethernet networks," IETF RFC 894, April 1984.

[10] Price et. al., "Efficient protocol independent compression," IETF Internet Draft, Feb. 2001.

[11] 정진욱, 변옥환, 이재광 공역, TCP/IP 네트워크, 진영사, pp.192, 1998.9A, pp.1583-1591, 2001.

[12] Diniel P. Bovet and Marco Cesati, Understanding the LINUX KERNEL, O'Reilly, 2001.

[13] A. Giovanardi, G. Mazzini, M. Rossi and M. Zorzi, "Improved header compression for TCP/IP over wireless links," Electronics Letters, vol.36, no.23, pp.1958-1959, Nov. 2000.

[14] G. Boggia, P. Camarda and V.G. Squeo, "ROHC+: a new header compression scheme for TCP streams in 3G wireless systems," in Proc IEEE Int. Conf. Communications, vol.5, pp. 3271-3278, 2002.

[15] L. Schwiebert, G. Richard, and Jiao. Changli, "Adaptive header compression for wireless networks," in Proc IEEE Int. conf. Local Computer Networks, pp.377-378, 2001.

[16] A. Kondoz, A. Sadka, S. Worrall, S. Fabri and A. Cellatoglu , "Robust header compression for real-time services in cellular networks," 3G Mobile Communication Technologies Int. Conf., pp. 124-128, 2001.

[17] W.S. Filippo, M.W Ritter, R.J Friday and A. Srivastava, A study of TCP performance over wireless data networks, IEEE VTC, vol.3, pp.2265-2269, 2001.

[18] K Svanbro, L.-E. Jonsson, H. Hannu and L.-A. Larzon, "Efficient transport of voice over IP over cellular links," IEEE GLOBECOM, vol.3, pp.1669-1676, 2000.

[19] Le. Khiem, C. Clanton, Liu. Zhigang and Zheng. Haihong, "Efficient and robust header compression for real-time services," IEEE WCNC, vol.2, pp.924-928, 2000.

[20] M. Degermark, L.-E. Jonsson, H. Hannu and K. Svanbro, "Wireless real-time IP services enabled by header compression," IEEE VTC, vol.2, pp.1150-1154, 2000.

[21] R. Magnus, U. Kunitz, M. Dziadzka, D. Verworner, M. Beck and H. Bohme, "Linux Kernel Internals," F1 Pub., 1999.

[22] Embedded Linux System, HyBus, 2003.

[23] Jon Crowcroft and Iain Phillips, TCP/IP

and Linux Protocol Implementation, Wiley, 2002.

저자소개

나종민(Jong-min Na)

충북대 전파공학과 석사 졸업
인소팩(주) 근무중
※관심분야 : 컴퓨터 네트워크, 헤더 압축

이종범(Jong-Beom Lee)

한밭대 전자공학과
충북대 전파공학과 석사 졸업
(주)코다컴테크놀러지 근무중

이인성(Insung Lee)

한국통신 연구개발단 전임연구원
Texas A&M Univ., Dept. of Electrical Eng.
ETRI 이동통신기술연구단 선임연구원
충북대학교 전기전자공학부 부교수
※관심분야 : 이동/위성통신 시스템, 신호처리

신병철(Byung-Cheol Shin)

KAIST 전자공학과 석사
KAIST 전자공학과 박사
KAIST 교수
충북대학교 교수
※관심분야 : 멀티미디어통신, 무선랜