리눅스에서의 분할패킷 재조합 성능 개선

변상익[0], 함유식, 김정인, 설순욱, 김명철
한국정보통신대학원대학교 공학부
{sibyun, hahm, ferma, suseol, mckim}@icu.ac.kr

Enhancement of IP Defragmentation in Linux

Sang-Ick Byun[0], Yu-Sik Hahm, Jeong-In Kim, Soon-uk Seol and Myungchul Kim
School of Engineering, Information and Communications University

요 약

컴퓨터에 있어서 네트워킹이 중요한 요소로 부각되면서 운영체제는 네트워킹을 보다 효율적으로 지원할 수 있어야 한다. 데이터가 송신자에서 수신자로 전송될 때, 데이터는 이동경로상의 라우터들을 경유하게 된다. 그러나 경유하는 중간회선과 라우터의 처리능력이 서로 다르며, 처리 가능한 크기보다 큰 패킷을 받을 경우는 적절한 크기로 분할되게 된다. 수신측에서는 분할된 패킷을 다시 재조합 하여 원래의 데이터로 복원시켜야 한다. 이러한 패킷 재조합은 운영체제의 커널에서 수행된다. 본 논문에서는 리눅스 커널에서의 분할패킷 재조합 과정을 개선함으로써 노드간 데이터 전송률을 향상시킬 수 있는 방안을 제시한다.

1. 서론

리눅스는 공개환경에서 개발되고 무료로 배포됨으로써 다양한 하드웨어를 지원하고 안정성을 확보하게 되면서 많은 사용자 층을 확보하게 되었다[1]. 리눅스의 네트워크 모듈은 소켓 인터페이스, 주소결정프로토콜(ARP), IP 라우팅, IP 계층으로 구성되어 있다[2]. 소켓 인터페이스는 다양한 네트워킹과 내부 프로세스간의 통신을 지원한다. ARP는 IP주소를 이더넷 주소와 같은 물리적 하드웨어 주소로 변환하는 역할을 한다. IP 라우팅은 특정한 IP 주소로 전송되는 IP 패킷이 어디로 보내져야 하는지를 결정하게 된다. 전송경로상의 중간 라우터들은 처리능력보다 큰 패킷을 처리해야 하는 경우, 라우터의 IP 계층에서는 패킷을 목적지까지 전달하기 위하여 최대 전송 단위(MTU)에 맞도록 패킷을 분할하게 된다. 그리고 분할된 패킷을 받은 수신측에서는 다시 재조합 하는 과정을 거쳐 원래의 패킷으로 복원한다. 분할된 패킷을 성공적으로 재조합 하기 위해서는 수신된 각각의 패킷의 순서를 정렬할 필요가 있다. 일반적인 네트워크 환경에서는 분할된 패킷의 처음 부분이 먼저 도착하고 마지막 부분이 나중에 도착하게 될 확률이 크다. 그러나, 현재의 리눅스 커널은 분할 패킷의 순서를 정렬하는 과정에서 링크드 리스트의 헤더부터 검색함으로써 분할 패킷의 재조합 과정에서 효율성을 감소시키고 있다. 본 논문에서는 리눅스의 자료구조와 검색순서를 개선함으로써 얻을 수 있는 성능향상 정도를 실험을 통하여 확인하였다. 본 논문의 구성은 다음과 같다. 2장에서 패킷 분할 및 재조합의 필요성과 리눅스에서의 구현에 대하여 분석한다. 3장에서는 패킷 재조합을 개선할 수 있는 개선된 방법을 제시하고 4장에서 실험을 통하여 성능 향상 정도를 측정한다. 마지막 5장에서 결론이 제시된다.

2. 배경

본 장에서는 먼저 패킷 분할에 대하여 설명하고 현재의 리눅스 커널에서 분할패킷이 재조합 되는 과정에 대하여 살펴본다.

2.1 패킷 분할

각각의 네트워크는 처리 가능한 최대 패킷의 크기를 규정하고 있다[3]. 이는 하드웨어의 처리능력, 운영체제, 프로토콜, 표준과의 호환성 등 복합적인 요인에 의하여 발생한다. 네트워크에서 처리 가능한 패킷 크기보다 큰 패킷을 보내야 할 경우 다음과 같은 해결방안을 고려할 수 있다. 하나의 해결방안은 라우팅을 수행함에 있어서 이러한 문제를 유발할 수 있는 네트워크로 패킷을 보내지 않도록 라우팅을 수행하는 방법이다. 그러나 이 방법은 마지막 수신측의 네트워크에서 패킷의 크기가 처리 가능한 크기를 넘을 경우에 패킷을 전송할 수 없는 문제점이 발생한다. 다른 방법으로 게이트웨이에서 패킷을 분할하고 각 분할패킷을 별도의 패킷으로 취급하는 것이다. 이 방법은 분할패킷을 수신한 후에 다시 원래의 패킷으로 재조합 하는 과정을 필요로 한다. 다음 절에서는 리눅스(kernel 2.2.12)에서 분할패킷이 어떤 과정을 거쳐 재조합 되는지 살펴본다.

2.2 리눅스의 분할패킷 재조합 과정

본 논문은 알짜리눅스6.1을 대상으로 하였으며, 커널 버전은 2.2.12 이다. 리눅스의 패킷 재조합 모듈은 *usr/src/linux/net/ipv4/ip_fragment.c*에 구현되어 있다. 그림1은 리눅스의 커널에서 IP 분할패킷 재조합 과정을 도식화 한 것이다.

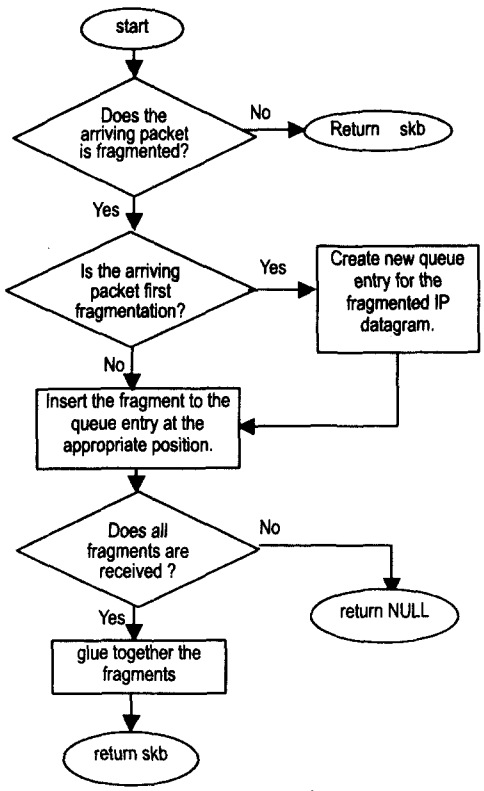


그림1. IP 분할패킷 재조합 흐름도

커널에서는 도착한 패킷이 분할되었는지 여부를 먼저 검사한다. 패킷이 분할되지 않은 경우에는 *skb* 포인터를 반환하고 패킷이 분할된 경우 그 분할패킷이 처음으로 도착한 분할패킷인지 검사한다. 처음 도착한 분할패킷인 경우 큐 엔트리를 생성하고 링크드 리스트로 구성된 분할패킷을 지시하도록 한다. 처음 도착한 분할패킷이 아닌 경우는 이미 큐엔트리가 존재하므로, 원래의 패킷순서에 맞도록 링크드 리스트의 적절한 위치에 삽입된다. 마지막으로 분할패킷이 모두 도착되었는지를 검사한 후 하나의 패킷으로 재조합 하고 패킷을 가리키는 *skb* 포인터를 반환함으로써 패킷 재조합 과정을 끝내게 된다.

패킷 재조합 과정에서 각 분할패킷의 위치를 정렬하는 과정을 큐엔트리를 중심으로 살펴보면 다음과 같다. 그림2는 첫번째 분할패킷이 도착했을 때 생성되는 큐엔트리의 구조를 보여주고 있다. 분할패킷이 링크드 리스트에 삽입되는 과정은 다음과 같다. 분할패킷이 삽입될 위치를 찾기 위해 **next*와 **prev* 두개의 포인터를 사용한다.

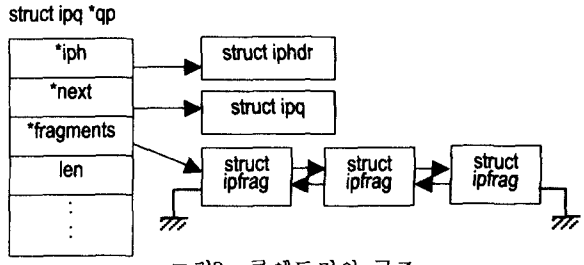


그림2. 큐엔트리의 구조

먼저 *prev*는 NULL 로 *next*는 첫번째 분할패킷(*qp->fragment*)을 가리키도록 초기화 한다. 새로 도착되는 분할패킷은 *next*가 가리키는 패킷과 offset 값을 비교하여 삽입될 위치를 찾게 된다. 이 방식에 의하면 새로 도착하는 모든 분할패킷은 항상 링크드 리스트의 첫번째 패킷부터 비교연산을 시작하여 적절한 위치를 찾을 때 까지 연산을 계속하게 된다. 따라서 원패킷의 끝에 위치하는 분할패킷의 경우 링크드 리스트의 모든 패킷들과 비교연산을 해야 한다. 일반적인 네트워크 환경에서는 원패킷의 끝부분에 위치하는 분할패킷일수록 수신측에 늦게 도착할 확률이 커지게 된다. 즉, 어떤 시점에서 수신측에 도착하는 분할패킷은 링크드 리스트의 끝에 삽입될 가능성이 가장 큼에도 불구하고 링크드 리스트의 처음부터 검색을 시작함으로써 검색 횟수가 많아지는 문제점이 발생하고 있다. 다음 장에서 이러한 문제점을 해결하기 위한 새로운 방법을 제시한다.

3. 패킷 재조합 성능개선을 위한 방법

검색횟수가 많아지는 문제점은 분할패킷이 삽입될 위치를 검색할 때 링크드 리스트의 끝부분부터 검색을 시작함으로써 개선될 수 있다. 이를 실제로 구현하기 위해서 큐엔트리의 구조와 프로그램 코드가 다음과 같이 수정되어야 한다.

그림3은 새로운 큐엔트리 구조를 보여준다.

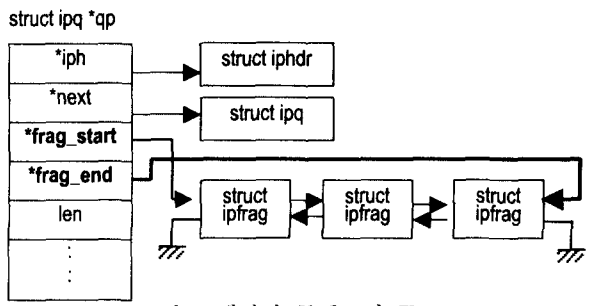


그림3. 개선된 큐엔트리 구조

새로운 큐엔트리에서 링크드 리스트의 첫번째 분할패킷을 지시하는 **fragments*는 **frag_start*로 재명명 되었으며, 링크드 리스트의 마지막 분할패킷을 지시하는 **frag_end* 포인터가 추가되었다. 그림4는 검색을 위한 기존의 리눅스커널 소스코드이며 그림5는 개선된 코드를 제시하고 있다.

```
if ((flags & IP_MF) == 0)     /* Is this the final fragment? */
        qp->len = end;
prev = NULL;
for(next = qp->fragments; next != NULL; next = next->next) {
        if (next->offset >= offset)     break;
        prev = next;
}
```

그림4. 기존의 검색코드

```
if ((flags & IP_MF) == 0){   /* Is this the final fragment? */
    qp->len = end;
    prev=qp->frag_end;
        next=NULL;
}
else {
    next = NULL;
     for(prev = qp->frag_end; prev != NULL; prev = prev->prev) {
        if (prev->offset <= offset) break;
        next = prev;
    }
}
```

그림5. 개선된 검색코드

개선된 코드의 분할패킷의 삽입위치를 검색하는 과정은 다음과 같다. 새로운 분할패킷이 도착했을 때 리눅스 커널에서는 More Field (*MF*)를 검사한다. 이 값이 0인 경우는 분할패킷이 원패킷의 마지막 부분임을 의미한다. 기존의 코드에서는 *MF*가 0인 패킷에 대하여 현재까지 도착한 모든 분할 패킷의 수 만큼 비교연산을 수행하지만 개선된 코드에서는 검색과정 없이 링크드 리스트의 마지막에 삽입한다. *MF*가 0이 아닌 경우에 검색은 링크드 리스트의 끝에서 시작하여 앞쪽으로 진행된다. 결과적으로 모든 패킷이 순서대로 도착한다는 가정 하에 n개의 분할패킷이 있을 때, 패킷 재조합은 기존의 방식이 n(n-1)/2회의 검색을 필요로 하지만 제안된 방식은 (n-2)회의 검색으로 가능하다.

4. 성능평가

본 장에서는 전송시간, 전송률의 측면에서 개선된 검색방법을 평가한다.

4.1 성능평가를 위한 시험환경

시험환경은 송신자, 수신자, 라우터로 구성하였다. 송신자와 라우터간의 연결은 기존에 사용중인 네트워크를 이용하였다. 송신자와 라우터간의 네트워크는 기존의 네트워크와 완전 분리된 별도의 서브넷을 구성하였다.

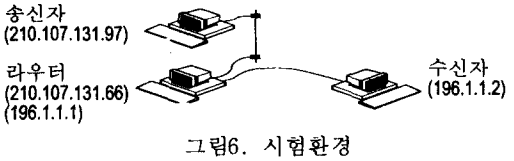
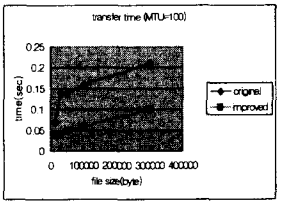


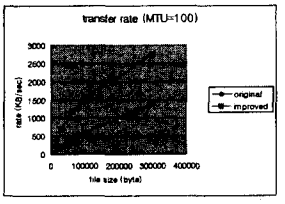
그림6. 시험환경

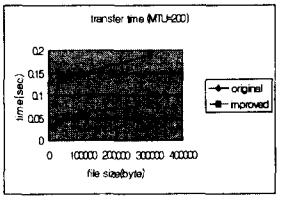
송신자와 라우터간의 네트워크는 이더넷으로 MTU의 크기는 기본값인 1500 바이트를 사용하였으며, 라우터와 수신자간의 네트워크에서는 다른 MTU 크기를 적용하여 실험결과를 도출하였다.

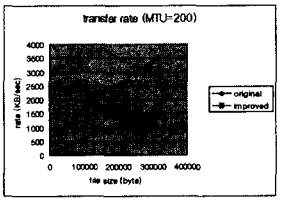
4.2 실험 결과 및 분석

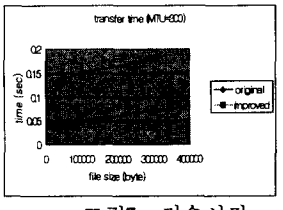
라우터에서 패킷 분할이 발생하도록 하기 위하여 MTU 크기를 100, 200, 300 바이트로 조정하고 각각의 경우에 대하여 파일의 전송시간, 전송률을 비교하였다. 그림7과 그림8은 전송되는 파일크기별 전송시간과 전송률을 보여주고 있다.











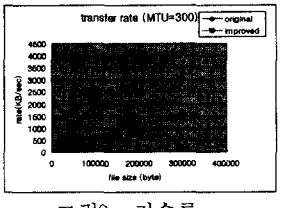


그림7. 전송시간 그림8. 전송률

위의 결과로부터 개선된 분할패킷 재조합 방법을 이용할 때 전송률이 최저 80%(MTU=100, File=15000) 에서 최고 165%(MTU=300, File=50000)의 성능향상을 보임을 알 수 있다. 즉, 분할패킷이 발생하는 모든 경우에 보다 짧은 전송시간과 높은 전송률을 제공함을 알 수 있다.

5. 결론

패킷 전송과정에서 중간 라우터의 처리능력보다 큰 패킷은 적절한 크기로 분할되고 각각의 패킷은 수신측에서 원래의 패킷으로 재조합된다. 본 논문에서는 기존의 리눅스 커널에서 분할패킷을 재조합하는 방법의 문제점을 제시하고, 이를 개선하기 위해 역방향 검색을 이용하여 새로운 커널을 구현하였다. 또한 개선된 방법을 적용하였을 때 전송률이 최저 80% 이상 향상됨을 실험을 통하여 확인하였다. 이로써 운영체제가 네트워킹을 효율적으로 지원하기 위해서는 수신측에 도착하는 각 패킷의 순서를 확률적으로 분석하고 커널구현에 적용하는 것이 매우 중요함을 알 수 있다.

참고문헌

[1] M Beck at el., Linux Kernel Internals, Addison Wesley, 1998.

[2] David A Rusling, **The Linux Kernel**, http://linux.flyduck.com/tlk/origtext/tlk-0.8-3.pdf.

[3] Andrew S. Tanenbaum, Computer Networks, Prentice-Hall, 1996.