비대칭 링크를 사용하는 홈 네트워크 게이트웨이에서 네트워크 성능 간섭 현상을 막기 위한 패킷 스케줄링 기법

(Preventing Network Performance Interference with ACK-Separation Queuing Mechanism in a Home Network Gateway using an Asymmetric Link)

홍 성 수 †

(Seongsoo Hong)

요 약 정보가전기기를 개발하는데 있어서 가장 어려운 일 중 하나는 네트워크 성능을 최적화하거나 성능상의 특이 현상을 극복하는 일이다. 본 논문에서는 상용 홈 네트워크 게이트웨이를 개발 하면서 이런 네트워크 성능 최적화의 한 문제를 발견하고 정의하며, 그 원인을 규명하고, 이를 해결한 연구를 기술한다. 본 논문의 홈 게이트웨이는 비대칭 링크인 ADSL 을 사용하는데 업링크로의 패킷 전송 속도가 다운링크로의 패킷 전송 유무에 간섭을 받는 특이 현상인 속도 간섭 현상을 보인다. 이러한 현상은 보통 receive livelock 문제로 생각되어 입력 큐에서의 패킷 경쟁이 그 원인이라고 분석된다. 그러나 본 논문에서는 실험적 검증을 통해 이 현상이 receive livelock과는 그 원인이 다른 새로운 문제임을 밝혀낸다. 추가적인 분석과 실험의 결과 출력 큐에서의 패킷 경쟁과 TCP 프로토콜의 메커니즘이 이 현상의 원인임을 밝힌다. 우리는 이 문제를 해결하기 위해 패킷 스케줄링 메커니즘을 제안하며 이를 홈 게이트웨이에 구현한다. 그 결과 개선된 홈 게이트웨이에서는 속도 간섭 현상이 완전히 해결됨을 보인다.

키워드 : 홈 네트워크 게이트웨이, 최적화, 성능 간섭, 비대칭 링크, 패킷 스케줄링

Abstract In development of network-enabled consumer electronics, much of the time and effort is spent analyzing and solving network performance problems. In this paper, we define an instance of such problems discovered while developing a commercial home network gateway. We then analyze its cause and propose a solution mechanism. Our home network gateway uses an asymmetric link (ADSL) and suffers from an undesirable phenomenon where downlink traffic interferes with upload speed. We call this phenomenon the network performance interference problem. While this problem can easily be confused with receive livelock caused by packet contention at the input queue, we find that this is not the case. By performing extensive experiments and analysis, we reveal that our problem is caused by packet contention at the output queue and certain intrinsic characteristics of TCP. We devise an ACK-separation queuing mechanism for this problem and implement it in the home network gateway. Our experiments show that it effectively solves the problem.

Key words : home network gateway, optimization, performance interference, asymmetric link, packet scheduling

1. 서 론

급격한 기술 융합의 결과로 최근의 정보가전기기(Consumer Electronics)는 다양하고 복합적인 기능을 제공하며, 다수의 통신 프로토콜을 사용하여 상호 연결되는 특징을 가지고 있다. 이런 기기에 탑재되는 소프트웨어는 자연히 많은 모듈들로 구성되며, 각 모듈들은 서로 밀접하게 상호 연관되어 동작한다. 이에 따라 정보가전기기용 소프트웨어를 단시간에 개발하고 최적화 하는 것은 매우 어려운 일이 되었다. 그 중에서도 네트워크 서브 시스템을 구현하는 소프트웨어를 개발하고, 그 성

† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
sshong@redwood.snu.ac.kr
논문접수 : 2005년 3월 18일
심사완료 : 2005년 10월 27일

능을 최적화 하는 것은 매우 힘든 작업이다. 그 이유는 네트워크 서브 시스템 자체가 매우 복잡하며, 성능을 최적화 하고자 할 때 여러 기기들간의 상호 연동으로 인한 효과를 예측하기 힘들며, 투여된 하드웨어 자원의 양과 성능이 비례하지 않는 특이 현상이 종종 발생하기 때문이다. 그 결과로, 정보가전기기의 개발 과정에서 초기 개발이 완료되었더라도 개발자들이 많은 시간과 노력을 추가로 투여하여야만 요구되는 네트워크 성능을 얻을 수 있는 일이 빈번하게 발생한다.

본 논문에서는 이런 고도의 복잡성을 지니는 대표적인 정보가전기기인 홈 네트워크 게이트웨이의 소프트웨어 성능 최적화 문제를 다룬다. (앞으로 간단히 홈 게이트웨이라고 부른다.) 홈 게이트웨이는 기본적으로 집 안의 많은 가전기기들이 인터넷에 연결될 수 있는 통로를 제공하는 기기이다. 아울러 방화벽, NAT, DHCP, SNMP와 같은 네트워크 서비스를 제공해 주는 동시에, 가전기기들에 대한 다양한 정보를 저장하고, 장치 드라이버를 제공하며, 웹 서버를 내장하여 사용자가 웹 브라우저를 통해 가전기기들을 원격 제어할 수 있는 인터페이스를 제공한다. 이처럼 홈 게이트웨이는 다수의 복합적인 기능을 동시에 제공하는 고도화된 정보가전기기이다.

홈 게이트웨이는 보통 양 방향 전송 속도가 다른 비대칭 링크인 ADSL(asymmetric digital subscriber line)을 통해 인터넷에 연결된다. ADSL은 이미 설치된 전화선을 이용하기 때문에 경제적으로 매우 저렴하면서도 기존의 전화선 모뎀에 비해 훨씬 빠른 성능을 가지고 있어 현재 많이 사용되고 있다. ADSL은 대부분의 인터넷 사용 패턴이 다운로드에 편중되어 있다는 것에 착안하여 다운로드 용으로 큰 대역폭을 할당하고 업로드 용으로는 적은 대역폭을 할당하여 제한된 대역폭으로 빠른 다운로드 속도를 실현한다.

그런데 홈 네트워크에 다양한 기기와 응용이 개발되면서, 최근의 인터넷 사용 패턴에서 업로드 트래픽이 점차 증가하고 있다. 예를 들어 인터넷 폰, 쌍방향 TV, 웹 카메라와 같은 가전기기들은 인터넷을 통해 영상이나 음성 데이타 등을 업로드 하는 경우가 많다. 또한 홈 게이트웨이에 연결되는 일반 PC에서도 과거에는 다운로드의 성능만이 중요하게 생각되었으나 P2P와 같은 새로운 영역의 통신 프로그램이 활발히 사용되면서 업로드의 양이 늘어나게 되었다. 이에 따라 엑세스 네트워크에서 업로드 성능이 점차 중요한 요소로 부각되고 있다.

우리는 이와 같은 특징을 가지는 상용 홈 게이트웨이를 개발하였다. 이 과정에서 개발 기간의 단축과 가격 경쟁력의 측면을 고려하여 COTS(common off the shelf) 소프트웨어인 리눅스 운영체제와 프로토콜 스택를 사용하였다. 그러나 이런 COTS 소프트웨어로 구성된 홈 게이트웨어의 프로토타입이 완성되었을 때, 실제 통신 대역폭이 가용 대역폭에 못 미치는 현상이 발견되었다. 우리는 이 문제가 ADSL과 같은 비대칭 통신 링크에서 한 쪽 방향의 네트워크 패킷 흐름이 다른 쪽 방향의 네트워크 패킷의 전송 속도를 저하시키는 현상이라고 밝히고, 이를 "비대칭 링크에서의 속도 간섭 문제"로 명명하였다. (앞으로 간단히 속도 간섭 문제라고 부른다.) 구체적으로 ADSL 링크를 사용하는 홈 게이트웨이에서 이 문제가 발생하면, 2.1 Mbps의 전송속도를 갖는 다운링크의 패킷의 영향으로 800 Kbps의 전송속도를 갖는 업링크의 속도가 저하되어, 전체 시스템은 허용될 수 없는 속도 저하를 경험하게 된다. 속도 간섭의 문제는 그 원인의 규명이 매우 어려운데, 본 논문에서는 이를 규명하고 그 해결책을 제안하여 구현과 성능 측정을 통해 검증하는 것을 다룬다.

앞에서 정의한 속도 간섭 문제와 유사하게 패킷 전송 요구량이 많아졌음에도 불구하고 실제 전송되는 양이 오히려 줄어드는 현상은 종종 receive livelock[1]으로 알려져 있다. 이 현상은 도착하는 모든 패킷들에 대해 하나의 FIFO 입력 큐를 할당하여 버퍼링하고, 인터럽트에 기반한 네트워크 패킷 처리를 하는 운영체제에서 빈번히 발생한다. 이러한 구조의 운영체제에서는 패킷이 입력 큐에 도착하는 속도가 패킷이 처리되는 속도보다 빠를 경우, 패킷들이 입력 큐에서 경쟁을 일으키고 결과적으로 드롭되어 전송량이 줄어들게 된다. 우리의 홈 게이트웨이의 운영체제로 사용된 리눅스를 비롯한 대부분의 운영체제들이 이러한 네트워크 패킷 처리 구조를 가지고 있다. 이 현상에 대한 기존의 연구 결과들을 응용하면 업링크와 다운링크 용으로 별도의 큐를 두고 이 큐들을 스케줄링 하도록 하여 업링크로의 패킷이 불공평하게 드롭되는 일을 방지할 수 있다. 그런데 예상과는 달리, 본 논문에서 구현과 실험을 통해 이러한 기법으로는 속도 간섭 문제를 해결할 수 없음을 보인다.

실제로 속도 간섭 문제는 receive livelock과는 매우 다른 문제이다. 우리는 실험과 분석을 통하여 이 문제의 원인이 입력 큐에서의 경쟁에 있지 않고, 출력 큐에서의 경쟁에 있으며, 동시에 TCP/IP 프로토콜의 특성과도 밀접하게 관련이 있음을 밝힌다. 이러한 분석을 바탕으로 우리는 출력 큐에서 TCP 프로토콜의 ACK 패킷을 우선적으로 처리하는 큐 관리 기법을 고안한다. 그리고 이 기법을 홈 게이트웨이의 리눅스 운영체제에 적용하기 위해 리눅스의 queuing discipline을 사용하여 구현하고 실험한다. 그 결과 속도 간섭 문제가 완전히 해결됨을 보인다.

비대칭 링크에서의 TCP 속도 문제는 많이 다루어져

왔다. 논문 [2]에서는 업로드 속도가 느린 비대칭 링크에서 다운로드 속도의 성능 문제를 다루었다. 논문 [3]은 [2]의 결과를 확장하여 양방향으로의 트래픽이 있을 경우의 성능 문제를 제기하고 그 해결책으로 각 TCP 연결 별로 별도의 큐를 두는 해결책을 제시했다. 논문 [4]는 ACK-filtering과 ACK-regeneration이라는 기법으로 양방향 트래픽의 성능 문제를 해결하고자 했다. 그러나 이들 논문들은 모두 실제 시스템이 아닌 시뮬레이션 만으로 검증된 것이라는 한계점을 가지고 있다. 또한 이들은 네트워크 종단의 PC나 서버의 패킷 처리 구조를 개선하여 문제를 해결하고 있다. 따라서 네트워크 중간에 설치된 홈 게이트웨이의 개선 만으로 성능 문제를 해결하여야 하는 본 논문과는 해결 방법의 적용 대상이 다르다. 또한 제시된 기법들은 업로드나 다운로드 중 한쪽 방향만의 속도 저하 문제를 해결하고 있어서 속도 간섭 문제의 완벽한 해결책이 되지 못하고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 이 논문의 대상이 되는 홈 네트워크 게이트웨이의 패킷 처리 방식과 네트워크 구성을 소개하고 속도 간섭 문제를 정의한다. 3장에서는 이 문제를 해결하기 위해 수행한 다양한 분석들을 설명한다. 4장에서는 출력 큐에서의 패킷 스케줄링 기법을 이용한 해결 방안을 소개한다. 그리고 이 기법을 실제 홈 게이트웨이에 구현한 내용과 그 결과를 기술한다. 마지막으로 5장에서 결론을 맺는다.

2. 문제 기술

본 장에서는 비대칭 링크에서의 네트워크 속도 간섭 문제를 구체적으로 설명한다. 그 전에, 앞으로 진행할 논의의 이해를 돕기 위해 먼저 홈 게이트웨이의 구성 요소와 패킷 처리 방식을 소개하고 속도 간섭 문제가 발생하는 네트워크 구성을 설명한다.

2.1 홈 게이트웨이의 구성 요소들과 패킷 처리 메커니즘

본 논문의 대상인 홈 게이트웨이는 표 1과 같은 하드웨어와 소프트웨어 구성 요소로 이루어져 있다. 메인 프로세서로는 ARM 아키텍처 기반의 SoC가 사용되며 런타임 메모리로는 8 Mbyte의 SDRAM, 프로그램 코드와 설정의 저장을 위해 2 Mbyte의 플래시 메모리가 사용된다. 메인 프로세서에 MMU가 없기 때문에 운영체제로는 가상 메모리가 없는 환경에서 동작하도록 수정된 리눅스인 uClinux가 사용된다. 이제 본 논문의 주요 논의 대상인 네트워크 부분을 중점적으로 설명하겠다.

홈 게이트웨이는 네트워크 하드웨어로 ADSL과 이더넷의 두 개의 인터페이스를 가진다. ADSL 인터페이스를 통해서는 인터넷과 연결되며, 이더넷 인터페이스를 통해서는 PC를 비롯한 홈 네트워크의 다양한 가전기기들과 연결된다. 네트워크 소프트웨어로는 RFC 2684, RFC 2364, RFC 1577의 세 가지 프로토콜 스택들이 있으며, 홈 게이트웨이의 동작 모드에 따라 이들 중 하나가 선택되어 사용된다. 이들은 기본적으로 각 네트워크 인터페이스로 도착한 패킷을 서로의 반대편 네트워크 인터페이스로 전송하는 일을 한다. 이들은 패킷의 헤더를 해석하고 조작하는 구체적인 방식에서 각각 차이점이 있지만, 이러한 차이점이 본 논문에서의 논의에 본질적인 영향을 끼치지는 않는다.

홈 게이트웨이의 네트워크 서브시스템의 구조와 패킷 처리 과정을 좀 더 자세히 설명하면 그림 1과 같다. 먼저 구조는 크게 1) 디바이스 드라이버, 2) 네트워크 프로토콜 스택, 3) 입/출력 큐의 세 가지 요소로 나눌 수 있다. 첫째, ADSL과 이더넷 장치를 구동하기 위한 각각의 네트워크 디바이스 드라이버들이 있다. 이 드라이버들은 네트워크 장치에 도착한 패킷을 메모리로 복사하고, 반대로 메모리에 저장된 패킷들을 네트워크 장치로 전송하는 일을 한다. 둘째, 네트워크 프로토콜 스택은 이렇게 메모리에 저장된 패킷들의 실질적인 처리를 담당한다. 구체적으로, 패킷의 헤더를 분석하고 라우팅 테이블을 참조하여 패킷이 전송되어야 할 네트워크 인터페이스를 결정하며, NAT등을 사용할 경우 헤더의 내용을 수정하는 작업 등을 수행한다. 처리가 끝나면 패킷은 출력을 위한 메모리에 저장된다. 셋째, 입/출력 큐는 디바이스 드라이버와 프로토콜 스택이 패킷을 주고 받기 위해 사용되는 in-memory 자료구조이다. 이 큐들은 FIFO방식으로 동작한다. 여기서 주목할 것은, 출력 큐

표 1 홈 네트워크 게이트웨이의 구성 요소들

하드웨어	메인 프로세서	ARM 아키텍처 기반의 SoC (MMU 없음, 속도: 40 MHz)
	네트워크 프로세서	ADSL DSP
	메모리	SDRAM: 8 Mbyte, 플래시 메모리: 2 Mbyte
	네트워크 인터페이스	ADSL × 1, 이더넷 × 1
소프트웨어	운영체제	uClinux[5] 2.4.17 (MMU가 없는 프로세서를 위한 리눅스)
	제공기능	웹 서버, NAT, NMP, 방화벽, IP 필터링, DHCP 등
	프로토콜 스택의 종류	• RFC 2684 (bridged Ethernet encapsulation) • RFC 2364 (PPP over AAL5 encapsulation) • RFC 1577 (IPv4 PDU encapsulation)

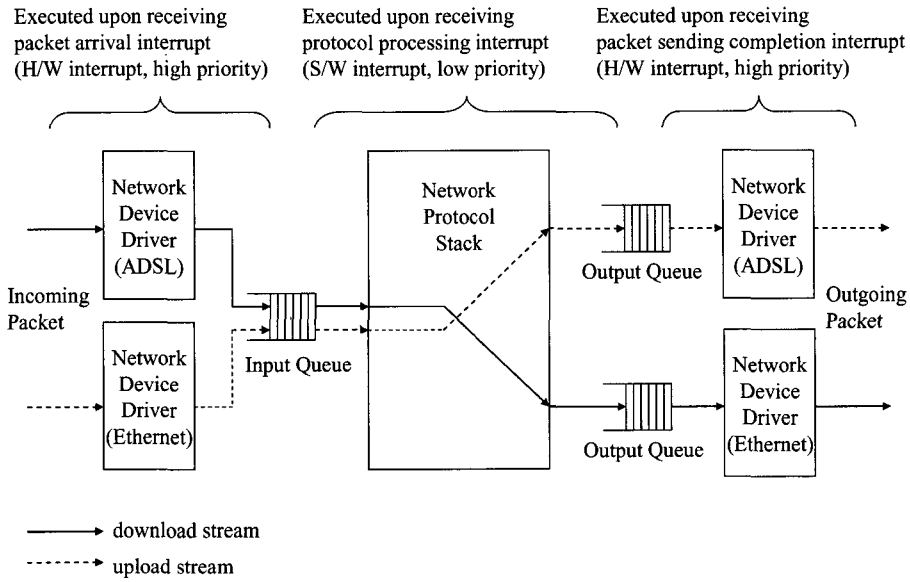

그림 1 홈 게이트웨이의 네트워크 서브시스템 구조와 패킷 처리 과정

는 각 디바이스 드라이버 별로 하나씩 존재하지만, 입력 큐는 모든 디바이스 드라이버에 대하여 단 하나의 큐가 존재한다는 점이다. 따라서 홈 게이트웨이로 도착한 모든 패킷은 이 단일 입력 큐에 저장되며 네트워크 프로토콜 스택에 의해 순서대로 하나씩 처리된다.

패킷은 크게 1) 패킷의 도착, 2) 프로토콜 처리, 3) 패킷 전송의 세 단계를 거쳐 처리된다. 각 단계는 패킷 도착 하드웨어 인터럽트 핸들러, 프로토콜 처리 소프트웨어 인터럽트 핸들러, 그리고 패킷 전송 완료 하드웨어 인터럽트 핸들러의 세 가지의 서로 다른 문맥에서 수행된다. 이 중에서 두 번째 단계는 소프트웨어 인터럽트 핸들러에 의해 수행되어서 가장 우선순위가 낮기 때문에 첫 번째와 세 번째 단계가 작동하지 않는 남은 시간에만 수행되며 언제든지 하드웨어 인터럽트가 발생하면 선점 당할 수 있다. 이 세 핸들러들은 각각 다른 길이와 주기로 수행될 수 있기 때문에 이 차이를 완화하기 위한 버퍼로 입/출력 큐가 사용된다. 그러나 이 큐들은 유한한 크기를 가지기 때문에 여유 공간이 없을 경우 패킷은 처리되지 못하고 자동적으로 폐기(drop)된다.

2.2 네트워크 구성

홈 게이트웨이가 사용되는 네트워크의 하드웨어와 소프트웨어 구성은 그림 2와 같다. 우선 하드웨어는 로컬 호스트, 홈 게이트웨이, 인터넷, 원격 호스트의 네 개의 노드들이 순서대로 연결된 형태로 구성되어 있다. 로컬 호스트는 이더넷 링크를 통해 홈 게이트웨이와 연결되며, 홈 게이트웨이는 다시 ADSL 링크를 통해 인터넷에 연결된다. 그리고 원격 호스트는 이더넷 링크로 인터넷과 연결되어 있다. 여기에서 인터넷은 사실 여러 개의 노드로 구성되어 있는 네트워크 이지만, 하나의 커다란 노드로 생각해도 본 논문을 이해하는 데에는 아무런 문

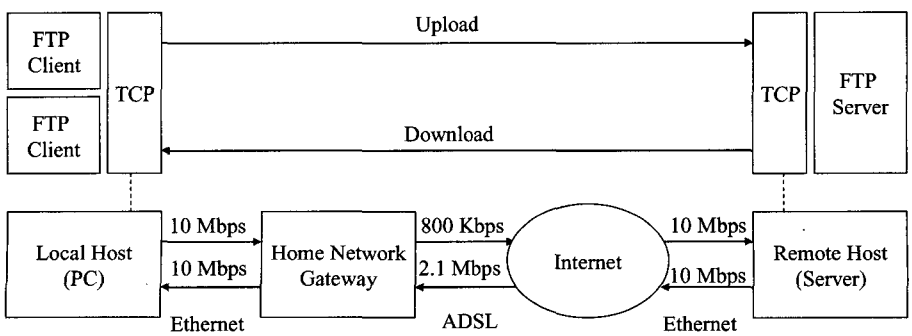

그림 2 속도 간섭 현상을 재현하기 위한 네트워크 하드웨어와 소프트웨어의 구성

제가 없다.

이더넷 링크와 ADSL 링크의 데이타 전송 속도에 관한 특성은 다음과 같다. 이더넷 링크는 대칭 링크이므로 양 방향의 전송 속도가 각각 10 Mbps로 동일하다. 반면, ADSL 링크는 양 방향의 전송 속도가 다른 비대칭 링크이다[6]. 홈 게이트웨이에서 인터넷으로의 업로드 속도는 약 800 Kbps이며, 인터넷에서 홈 게이트웨이로의 다운로드 속도는 이보다 약 2.6배 빠른 2.1 Mbps이다. 이 속도는 인터넷 서비스 사업자가 제공하는 ADSL Lite 서비스에 해당하는 성능으로 ADSL Lite는 가장 널리 사용되는 ADSL 서비스 중 하나이다.

다음으로 네트워크 소프트웨어는 다음과 같이 구성되어 있다. 우리는 속도 간섭 현상의 원인을 분석하기 위해 의도적으로, FTP를 사용하는 간단한 구성을 택한다. 이 구성에서 원격 호스트에는 멀티스레드 FTP 서버가 수행되며, 로컬 호스트에는 두 개의 FTP 클라이언트 프로그램이 수행된다. 이 클라이언트와 서버 프로그램은 TCP 프로토콜을 사용하여 데이타를 주고받는다. 두 클라이언트 중, 한 클라이언트는 서버에게 파일을 전송하며, 다른 클라이언트는 서버로부터 파일을 전송받는다. 전송에 사용되는 파일은 100 Mbyte이상의 큰 크기의 임의의 파일로, 각 클라이언트는 이 파일들을 끊임없이 재 전송한다. 이러한 소프트웨어 구성은 비록 간단하지만, P2P 응용 프로그램이나 인터넷 전화와 같이 동시에 양방향으로 끊임없이 데이타를 전송하는 네트워크 응용의 특성을 잘 반영하고 있기 때문에 이러한 구성을 사용하는 것이 본 논문의 일반성을 저하시키지는 않는다.

2.3 네트워크 성능 간섭 현상

네트워크 성능 간섭 현상은, 이렇게 구성된 홈 게이트웨이에서 양 방향으로 동시에 데이타를 전송할 경우, 업로드 속도가 급격히 줄어드는 현상으로 정의된다. 이 현상을 명확히 규명하기 위해 1) 업로드 방향으로만 파일을 전송하는 경우, 2) 다운로드 방향으로만 파일을 전송하는 경우, 그리고 3) 양방향으로 파일을 전송하는 경우의 세 가지에 대하여 파일 전송 속도를 측정하여 그림 3과 같은 결과를 얻었다. 이 그림은 업로드와 다운로드 방향으로의 파일 전송을 켜고 끄는 각 조합에 따른 업로드와 다운로드 속도의 변화를 보여주고 있다. 전송 속도는 파일의 실효 전송량을 기준으로 측정되었다. 그림 3에서 볼 수 있듯이 양방향으로 파일을 전송할 경우, 성능 간섭 현상이 발생하여 업로드의 속도가 750 Kbps에서 250 Kbps로 큰 폭(약 67%) 감소한다.

1) 업로드 방향으로만 파일을 전송할 경우: 약 750 Kbps
2) 다운로드 방향으로만 파일을 전송할 경우: 약 1900 Kbps
3) 양방향으로 파일을 전송할 경우: 업로드: 약 250 Kbps, 다운로드: 약 1900 Kbps

이 현상은 홈 게이트웨이를 실제 가정에서 사용하는데 있어서 다양한 문제점을 일으킬 수 있다. 앞서 언급했듯이 이 현상은 P2P프로그램은 물론 홈 네트워크 장치들의 성능을 크게 저하시킨다. 예를 들어 홈 게이트웨이에 PC와 집안을 감시하기 위한 네트워크 비디오 카메라가 함께 연결되어 있다고 하자. 그리고 PC에서는 현재 대용량의 미디어 파일을 다운로드 받고 있다고 하자. 만약 사용자가 외부에서 비디오 카메라에 원격 접속해서 집 안의 상황을 살펴보고자 할 경우, 비디오 카메라는 사용자의 원격 단말기로 영상 데이타를 업로드하게 된다. 그러나 간섭 현상 때문에, 영상 데이타의 전송 속도가 당초 기대되었던 것보다 급격히 감소하여 집 안의 상황을 원활히 살펴볼 수가 없게 된다. 사용자가 이 문제를 해결하기 위해서 PC에 접속하여 다운로드를 잠시 중지할 수 밖에 없는데 이는 분명히 매우 불편하며 임시 방편의 해결책이다. 홈 게이트웨이는 이러한 상황이 일어나지 않도록 네트워크 패킷의 흐름을 효과적으로 관리할 수 있어야만 한다.

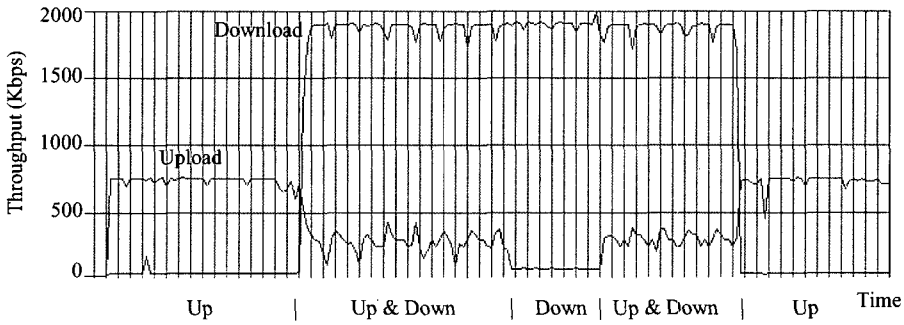

그림 3 업로드/다운로드 방향으로의 파일 전송 유무의 조합에 따른 각 방향의 파일 전송 속도. Up과 Down은 각각 업로드와 다운로드 방향으로 파일을 전송하였음을 나타낸다.

3. 입력 큐와 출력 큐에서의 패킷 경쟁 분석

본 장에서는 속도 간섭 문제의 원인을 분석하고, 실험을 통해 이를 검증한다. 우선 본격적으로 원인을 알아보기 전에 1) ADSL 링크가 물리적으로 동시에 양 방향 최고 속도를 지원할 수 있는지의 여부를 실험을 통해 알아본다. 2) 그런 다음, 속도 간섭 문제와 TCP 프로토콜과의 연관성을 밝히기 위해 TCP의 동작 메커니즘에 대해 설명한다. 이에 따르면 속도가 감소하는 것은 입력 큐 혹은 출력 큐에서의 패킷 경쟁으로 설명된다. 3) 먼저 입력 큐에서의 패킷 경쟁에 대해 분석해 본다. 입력 큐에서의 패킷 경쟁은 널리 알려진 receive livelock 문제의 원인이기도 하다. 우리는 여러 개의 입력 큐를 두고 큐 스케줄링 기법을 적용하도록 하는 receive livelock 문제의 해결 방법을 적용하여 본다. 그러나 이 방법으로는 속도 간섭 문제가 해결되지 않는다는 것을 실험을 통해 밝힌다. 4) 다음으로 출력 큐에서의 패킷 경쟁에 대해 분석한다. 그 결과 속도 간섭 문제의 원인은 업로드 방향으로 전송되는 ACK 패킷과 데이타 패킷이 ADSL 링크를 향한 출력 큐에서 서로 경쟁하기 때문임을 밝힌다.

3.1 달성 가능한 대역폭 분석

본 절에서는 본격적인 원인 분석에 들어가기에 앞서, 근본적으로 속도 간섭 문제가 해결 가능한 문제인지를 알아본다. 만약 ADSL 링크에서 양 방향으로 동시에 최대 속도를 달성하는 것이 물리적으로 불가능 하다고 하면, 어떠한 소프트웨어적인 방법을 사용하여도 이 문제를 해결할 수 없을 것이기 때문이다.

이 여부를 확인하기 위해 그림 4와 같이 네트워크 소프트웨어를 구성하였다. FTP 서버와 클라이언트 대신에 Gen과 Rcv라는 프로그램이 각각 로컬 호스트와 원격 호스트에 쌍을 이루어 설치된다. Gen 프로그램은 정해진 속도로 UDP 패킷을 생성하는 프로그램이다. 생성된 UDP 패킷은 TCP의 데이타 패킷의 크기와 같은 1500 byte의 크기를 가지며, 순서 번호가 기록된다. Rcv프로그램은 반대로 Gen이 생성한 UDP 패킷을 받는 프로그램이다. Rcv는 받은 UDP 패킷에 기록된 순서 번호를 출력하고, 받은 총 패킷의 개수와 전송 속도를 출력한다. 이렇게 소프트웨어를 설치한 뒤, 로컬 호스트와 원격 호스트에 설치된 Gen 프로그램이 반대편 호스트의 Rcv 프로그램으로 각각 링크의 최대 속도인 800 Kbps, 2.1 Mbps의 속도로 UDP 패킷을 전송하도록 한다. 이 때, 각 호스트의 Rcv 프로그램이 얼마만큼의 UDP 패킷을 받았는지 측정한다.

측정 결과 모든 패킷이 전혀 손실되지 않고 전송되었음을 확인하였다. 이를 통해 물리적으로 는 한 쪽 방향의 패킷이 다른 쪽 방향의 패킷 전송을 방해하지는 않는다는 것을 알 수 있었다. 이러한 점들을 고려해 볼 때, 속도 간섭 문제는 ADSL링크의 물리적인 한계 때문이 아니며 소프트웨어에 의해 발생된 문제라고 결론지을 수 있다.

3.2 TCP의 작동 원리

위의 분석에서UDP 프로토콜을 사용하였을 때에는 속도 간섭 문제가 일어나지 않았기 때문에, 이 문제는 TCP 프로토콜[7]의 특성과 관련이 깊다고 분석된다. 따라서 본 절에서는 앞으로의 논의를 위해 TCP 프로토콜의 동작 메커니즘에 대해 간단히 설명한다.

TCP 프로토콜은 데이타 패킷을 전송한 후, 이에 대한 ACK(acknowledge) 패킷이 도착하면 다음 데이타 패킷을 전송하는 것을 기본으로 하고 있다. 그러나 보통은 전송 효율을 높이기 위해, ACK 패킷을 기다리는 동안 *wnd* 개의 데이타 패킷들을 연속하여 보내도록 하고 있다. 이를 초과하여 데이타 패킷을 보내고자 할 때에만 ACK 패킷을 기다리게 된다.

이 *wnd*는 수행 중에 바뀔 수 있는 값이며 *maxwnd*와 *cwnd*라는 두 개의 변수 중 작은 값으로 결정된다. 먼저 *maxwnd*는 *wnd*가 가질 수 있는 최대 값, 즉 전송 버퍼의 크기이다. 이 값은 운영체제의 종류에 따라 다르지만 보통 20~100 정도로 고정되어 있으며, 본 논문의 실험 환경에서는 88로 고정되어 있다. 반면, *cwnd*는 1부터 시작하여 ACK 패킷을 받음에 따라 1씩 증가하는 값이다. 이 *cwnd*가 증가하면서 TCP는 점점 빠른 속도로 데이타를 보내게 된다. 이러한 도중에 타이머 종료 등을 통해 데이타 패킷이 유실된 것을 발견하면 *cwnd*

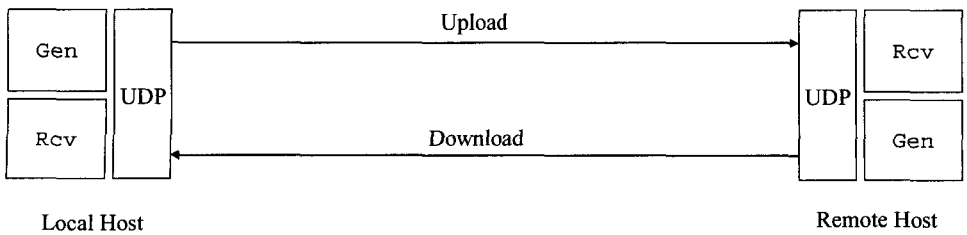

그림 4 양방향으로 UDP 패킷을 전송하는 소프트웨어 구성

를 1로 감소시켜 다시 느린 속도로 패킷을 보내기 시작한다.

이상을 종합하면 TCP에서는 다음의 두 가지 경우가 발생할 경우 전송 속도가 감소한다.

- ACK 패킷이 과도하게 지연되는 경우이다. *wndmax* 개의 데이타 패킷을 보낸 후에도 ACK 패킷이 도착하지 않으면 그 패킷이 도착할 때까지 아무런 데이타 패킷을 보내지 못하고 기다려야만 한다. 따라서 단위 시간 동안 보내는 평균 데이타 양이 감소한다.
- 데이타 패킷이 유실되는 경우이다. 앞서 설명하였듯이 *cwnd*를 1로 감소시켜 다시 느린 속도로 패킷을 보내기 시작한다.

3.3 입력 큐에서의 패킷 경쟁

TCP의 전송 속도가 감소하는 경우는, ACK 패킷이 지연되는 경우와 데이타 패킷이 유실되는 경우의 두 가지임을 설명했다. 이 두 경우 모두 큐에서 패킷 경쟁이 있을 때 발생한다. 홈 게이트웨이에는 입력 큐와 출력 큐의 두 출력 큐가 있다. 본 절에서는 먼저 입력 큐에서의 패킷 경쟁에 대해 분석한다.

입력 경쟁에서의 패킷 경쟁은 receive livelock[1] 문제와 관련이 깊다. 이 문제는 속도 간섭 문제와 같이, 패킷 전송 요구량이 많아졌음에도 불구하고 실제 전송되는 양이 오히려 줄어드는 현상으로 정의된다. 이 현상은 하나의 FIFO 입력 큐를 할당하고, 도착하는 모든 패킷들을 이 큐에 버퍼링하며, 인터럽트에 기반한 네트워크 패킷 처리를 하는 운영체제에서 빈번히 발생한다. 이러한 구조의 운영체제에서는 패킷이 입력 큐에 도착하는 속도가 패킷이 처리되는 속도보다 빠를 경우, 패킷들이 입력 큐에서 경쟁을 일으키고 결과적으로 폐기되어 전송량이 줄어들게 된다. 우리의 홈 게이트웨이의 운영체제로 사용된 리눅스를 비롯한 대부분의 운영체제들이 이러한 네트워크 패킷 처리 구조를 가지고 있다.

홈 게이트웨이에서도 이렇게 입력 큐에서 패킷 경쟁이 일어날 수 있다. 홈 게이트웨이는 표 1에서 볼 수 있듯이 가격 경쟁력을 높이기 위해 40 MHz의 느리고 값싼 프로세서를 사용하며, 복잡한 프로토콜 스택을 거치기 때문에 패킷 처리 속도가 느린 편이다. 이렇게 경쟁이 일어나면 느린 속도로 도착하는 업로드 패킷은 한정된 입력 큐의 공간을 차지하기 위한 경쟁에서 다운로드 패킷에게 속도가 느린 비율만큼 지게 되어 다운로드 패킷보다 훨씬 더 빈번히 폐기된다. 다운로드 패킷도 폐기되지만 그 비율이 크지는 않다. 이처럼 업로드 방향의 패킷이 폐기되면 업로드 속도가 저하된다. 왜냐하면 전송 프로토콜로 사용된 TCP에서는 타이머 등을 통해서 데이타 패킷이 도중에 유실되었음을 알게 되면, 일정한 시간 내에 보낼 수 있는 데이타 패킷의 최대 개수(congestion window 라고 부른다)를 줄여 전송 속도를 낮추기 때문이다[7].

이런 문제는 보통 1) 각 네트워크 인터페이스 마다 별도의 입력 큐를 두고 2) 큐 스케줄링 메커니즘을 사용하여 사용자가 정한 비율대로 각 큐가 선택되도록 하는 class-based queuing 으로 해결한다. 첫째, 별도의 입력 큐를 가지고 있기 때문에, 반대 방향의 패킷 흐름 때문에 입력 큐에 남은 공간이 없어져서 패킷이 폐기되는 경우가 방지된다. 둘째, 큐 스케줄링 메커니즘은 패킷 한 개를 처리하기 위해, 여러 개의 입력 큐들 중에서 패킷을 가져올 큐를 공평하게 선택하기 위해 사용된다.

그림 5는 이 방법이 실제로 속도 간섭 현상을 해결하는지 확인하기 위해 홈 게이트웨이에 적용된 모습이다. 각 입력 큐의 크기는 100으로 설정되어 있으며, f와 g값을 조정하여 각 큐가 선택되는 비율을 조절할 수 있도록 되어 있다. 큐 스케줄링 알고리즘으로는 다양한 알고리즘들을 사용할 수 있는데 우리는 credit/debit 알고리즘[8]을 사용한다. 이 알고리즘은 몇 개의 정수 연산으로 구현 가능한 간단한 구조를 가지고 있어서, 문제 해결 여부를 빠르게 확인하는데 적합하여 선택되었다. Credit/debit 알고리즘의 구체적인 작동 방식에 대한 설명은 논의의 범위를 벗어나므로 생략한다.

그러나 실험 결과 이 기법으로는 속도 간섭 문제를 해결할 수 없었다. 각 입력 큐의 크기와 f와 g의 값을 다양한 값으로 변화시켜 보았지만 업로드 전송 속도는 여전히 약 250 Kbps로 개선되지 않았다. 이 결과는 입력 큐에서 패킷 경쟁이 발생하지 않음을 암시한다. 이를 실제로 확인하기 위하여 홈 게이트웨이가 동작 중일 때 입력 큐에 저장된 패킷의 개수를 측정하였다. 그 결과 그 개수는 0-5개에 불과하여 거의 항상 비어있어서 경쟁이 발생하지 않음을 확인했다.

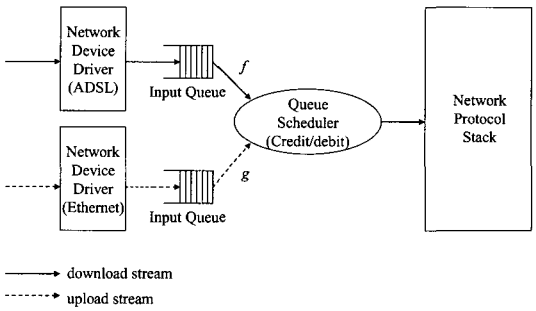

그림 5 별도의 입력 큐와 큐 스케줄러로 구성된 홈 게이트웨이의 네트워크 서브시스템 구조. 출력 큐 이후의 구조는 생략되어 있음. 정수 f와 g는 각각 ADSL과 이더넷 드라이버의 입력 큐를 선택할 비율임

결과적으로 속도 간섭 문제는 기존의 고속, 대용량 네트워크 서버나 라우터에서는 쉽게 관찰되었던 receive livelock 문제와 그 현상이 비슷하기는 하나 원인은 전혀 다르다. 이는 양 방향의 전송 속도가 비대칭인 ADSL을 사용하는 홈 게이트웨이의 특징으로 인하여 새롭게 발생한 문제이다. 따라서 이 문제는 기존에 널리 알려진 대로 입력 큐에서의 경쟁을 회피하는 방식으로는 해결될 수 없다.

3.4 출력 큐에서의 패킷 경쟁

입력 큐에서의 경쟁은 없다는 것이 밝혀졌으므로, 간섭에 의한 네트워크 성능 저하가 있다면 출력 큐에서 경쟁이 발생하였음을 알 수 있다. 본 절에서는 홈 게이트웨이의 업링크와 다운링크로 향하는 두 개의 출력 큐에서의 패킷 경쟁을 분석한다. 먼저 업링크로 향하는 출력 큐에서는 실제로 경쟁이 일어나서 ACK 패킷과 데이타 패킷이 유실됨을 보인다. 그리고 이러한 패킷 유실이 각각 업로드와 다운로드 속도에 어떠한 영향을 미치는지 설명한다. 반면 다운링크로 향하는 출력 큐에서는 경쟁이 일어나지 않아 이 큐가 성능에 미치는 영향은 없음을 보인다.

먼저 업링크 방향의 출력 큐를 분석한다. 이 출력 큐에는 업로드 트래픽에 대한 데이타 패킷들과, 다운로드 트래픽에 대한 ACK 패킷들이 같이 존재한다. 이 패킷들은 10 Mbps속도의 이더넷 링크에서 도착하여 800 Kbps 속도의 ADSL 링크로 전송되기 때문에 출력 큐에 쌓이고 서로 경쟁한다. 앞서 언급했듯이 로컬 호스트와 원격 호스트는 각각 최대 88개의 데이타 패킷들을 연속하여 전송할 수 있다. 또한 로컬 호스트는 원격 호스트에게서 88개의 데이타 패킷들을 받으면 이들에 대한 88개의 ACK 패킷들을 업로드 방향으로 연속하여 전송한다. 따라서 업로드 링크에는 최대 88개의 데이타와 88개의 ACK 패킷들이 존재하게 된다. 그런데 출력 큐는 100의 크기를 가지고 있기 때문에 이 패킷들의 일부는 저장되지 못하고 유실된다. 이제 이 경쟁 과정에서 ACK 패킷과 데이타 패킷들이 유실될 때 이것이 업로드와 다운로드 성능에 어떠한 영향을 미치는지 살펴본다.

먼저, 다운로드 트래픽에 대한 ACK 패킷이 유실되는 것은 다운로드 속도에 큰 영향을 미치지 않는다. 그 이유는 다음과 같다. ACK 패킷은 cumulative한 특성을 가지고 있어서 몇 개의 ACK 패킷이 유실되었더라도 그 이후에 최소한 하나의 ACK 패킷을 받기만 하면 모든 데이타 패킷이 잘 전송되었음을 나타낸다. 따라서 모든 ACK 패킷이 유실되지 않는 한 원격 호스트는 데이타 패킷을 보내는 속도를 감소시키지 않는다.

반면 업로드 트래픽에 대한 데이타 패킷은 하나라도 유실되면 업로드 속도를 감소시킨다. 이 경우 로컬 호스트는 앞서 설명했듯이 *cwnd*를 1로 감소시키고 느린 속도로 패킷을 보내기 시작한다. 이러한 추세가 계속되면 업링크 방향의 출력 큐는 그림 6과 같은 상태가 된다. 상대적으로 고속인 다운로드 속도는 유지되기 때문에 항상 약 88개의 ACK 패킷이 출력 큐에 존재하게 된다. 따라서 출력 큐에는 약 12개의 작은 여유 공간만이 있다. 로컬 호스트는 *cwnd*를 증가시키면서 업로드 속도를 증가시키려 하지만, 곧 여유 공간이 부족하여 데이타 패킷이 유실되고 다시 *cwnd*를 1로 감소시킨다. 따라서 업로드 속도는 증가와 감소를 반복하여 평균적으로는 낮은 속도를 유지한다. 이것이 바로 속도 간섭 현상의 원인이다.

이제 다운링크 방향의 출력 큐의 경우를 살펴본다. 이 큐에서는 2.1 Mbps의 속도로 패킷이 들어오고 10 Mbps의 속도로 패킷이 떠나므로 경쟁이 발생하지 않는다.

홈 게이트웨이 내부의 다운링크 방향의 출력 큐에서 패킷 경쟁이 발생하지 않는다고 원격 호스트에서 시작하여 로컬 호스트에 이르는 다운링크 전체 경로에서 패

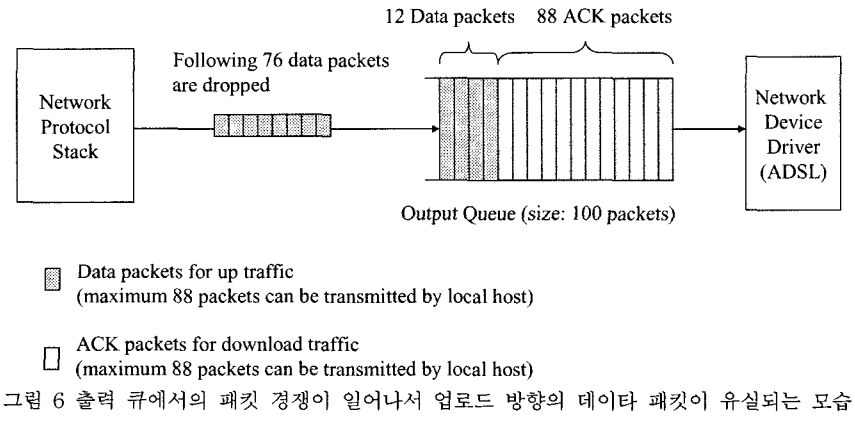

그림 6 출력 큐에서의 패킷 경쟁이 일어나서 업로드 방향의 데이타 패킷이 유실되는 모습

킷 경쟁이 발생하지 않는 것은 아니다. 패킷 경쟁이 발생할 수 있는 곳은 홈 게이트웨이의 내부가 아닌 인터넷에서 ADSL 링크 방향으로의 출력 큐이다. 여기에서는 패킷이 들어오는 속도는 10 Mbps이며, 떠나는 속도는 2.1 Mbps이기 때문이다. 만약 이 출력 큐의 크기가 홈 게이트웨이에서처럼 양 방향의 모든 패킷들을 저장하지 못할 정도로 작다면 위에서 설명한 반대 현상이 일어나 다운로드 방향의 데이타 패킷이 유실되어 다운로드 속도 또한 감소하게 된다. 그러나 인터넷을 구성하는 라우터나 게이트웨이들은 burst 트래픽을 감당하기 위해 매우 큰 출력 큐를 가지는 경우가 대부분이므로 실제 다운로드 속도가 감소하는 경우는 발생하지 않는다.

4. 개선된 출력 큐와 구현 결과

우리는 3장에서 속도 간섭 현상의 원인은 업링크 방향으로의 출력 큐에서 업로드 트래픽의 데이타 패킷이 유실되기 때문임을 분석했다. 본 장에서는 이 현상을 막기 위해 개선된, 출력 큐의 구조와 동작 방식에 대해 설명한다. 그리고 개발된 홈 게이트웨이에 리눅스의 queuing discipline을 이용하여 개선된 출력 큐를 구현하고 적용한 내용을 설명한다. 마지막으로 실험을 통해 개선된 출력 큐가 속도 간섭 문제를 효과적으로 해결함을 보인다.

4.1 개선된 출력 큐 메커니즘

개선된 출력 큐는 크게 1) 출력 큐의 크기 증가, 2) ACK 패킷을 우선 처리하는 패킷 스케줄링의 두 가지를 핵심 아이디어로 한다. 이제 이 두 핵심 아이디어들을 자세히 설명한다.

첫째, 출력 큐의 크기가 증가된다. 업링크에는 최대 176(= 88 + 88)개의 ACK과 데이타 패킷들이 존재할 수 있지만 출력 큐의 크기는 100으로 모든 패킷들을 저장할 수 없었기 때문에 데이타 패킷의 유실이 발생했다. (ACK 패킷도 유실되지만 이는 성능에 영향을 끼치지 않는다.) 따라서 데이타 패킷 유실을 막기 위해서는 출력 큐의 크기를 176 이상으로 크게 하는 것이 필요하다. 그런데 이렇게 큐 크기를 크게 하면 다운로드 트래픽에 대한 ACK 패킷이 과도하게 지연되어, 지금까지와는 반대로 다운로드 속도가 감소하게 된다. ACK 패킷이 과도하게 지연되는 이유는 다음과 같다. TCP 프로토콜은 88개의 데이타 패킷들을 연속하여 보내기 때문에 출력 큐에는 이 데이타 패킷들이 연속적으로 저장되어 있으며 ACK 패킷은 이 데이타 패킷들 뒤에 저장된다. 그래서 ACK 패킷들은 큐에 삽입된 후 전송되기 위해 빠져나오기까지, 이 88개의 데이타 패킷들이 모두 큐에서 전송되는데 걸리는 시간만큼 지연된다.

둘째, 이렇게 ACK 패킷의 과도한 지연을 막기 위해, ACK 패킷을 우선 처리하는 패킷 스케줄링 기법이 적용된다. 그림 7은 이렇게 ACK 패킷을 우선 처리하기 위해 개선된 출력 큐의 구조이다. 개선된 출력 큐는 layer 3인 IP 스택(네트워크 프로토콜 스택의 가장 하위 스택)과 layer 2인 디바이스 드라이버 사이에 존재한다. 이는 패킷 분류기, ACK 패킷들을 위한 별도의 출력 큐, 패킷 스케줄러로 구성된다. 패킷 분류기는 네트워크 프로토콜 스택에서 처리가 끝난 패킷들을 ACK 패킷과 그 외의 패킷들로 분류한다. 분류된 ACK 패킷은 ACK 패킷만을 위한 별도의 출력 큐에 저장된다. ACK가 아닌 패킷은 기존의 출력 큐에 저장된다. 패킷 스케줄러는 ACK 패킷을 위한 출력 큐를 먼저 검사하여 이 큐에 ACK 패킷이 존재한다면 그 패킷을 가져온다. 이 큐에 아무런 패킷도 없을 경우에만 기존의 출력 큐에서 패킷을 가져온다.

이와 같이 ACK 패킷에 무조건적인 우선순위를 주면

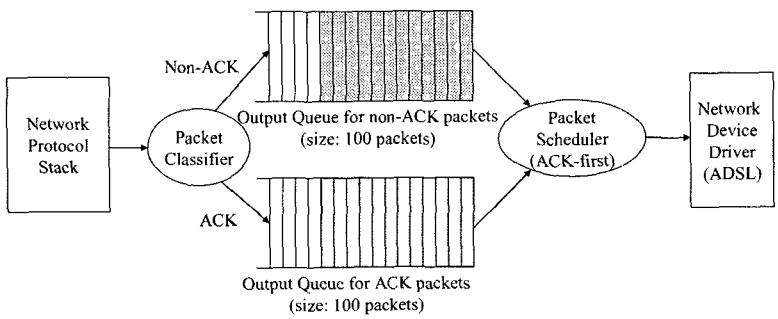

그림 7 속도 간섭 현상을 해결하기 위한 출력 큐의 구조

데이타 패킷들이 전송되지 못하고 고착상태에 빠질 수도 있다는 것을 고려해야 한다. 그러나 우리의 네트워크 구성에서는 ACK 패킷들을 전송하기 위해 필요한 대역폭이 데이타 패킷들을 전송하기 위해 필요한 대역폭보다 훨씬 작으므로 그러한 고착상태가 발생하지 않는다. 이는 다음과 같이 설명된다. 데이타 패킷 하나에 ACK 패킷 하나가 생성되므로, 업링크로 전송되는 ACK 패킷의 초당 개수는 다운링크로 전송되는 데이타 패킷의 초당 개수인 179.2 packets/s (= 2.1 Mbps / 1500 byte)와 같다. ACK 패킷의 크기는 66 byte이므로 179.2 packets/s의 속도로 전송되면 약 92 Kbps (= 66 byte × 179.2 packets/s)의 대역폭을 사용하게 된다. 이는 업링크 대역폭인 800 Kbps의 11.5%에 해당한다. 따라서 ACK 패킷에 우선순위를 주어도 여전히 데이타 패킷들이 전송될 충분한 대역폭이 남아있다.

이 경우 업로드와 다운로드 트래픽의 지연 시간에 대해서 분석을 하면 다음과 같다. 첫째, 다운로드 트래픽의 지연 시간은 기존에 비하여 오히려 줄어든다. 다운로드 트래픽을 위한 ACK 패킷을 최우선 순위로 처리하기 때문이다. 둘째, 업로드 트래픽의 지연 시간은 기존에 비하여 증가한다. 최악의 경우 업로드 트래픽을 위한 데이타 패킷은 다운로드 트래픽을 위한 ACK 패킷들 88개가 모두 전송을 마칠 때까지 기다려야 한다. 이 시간을 계산하면 58 ms (= 66 byte × 88 / 800 Kbps)가 된다. 반면 평균적인 경우 업로드 데이타 패킷은 약 11.5%에 해당하는 지연 시간 증가를 겪는다. 위에서 설명하였듯이 업링크로 전송되는 ACK 패킷들은 업링크 대역폭의 약 11.5%를 사용하기 때문에 업로드 데이타 패킷의 입장에서는 업링크 대역폭이 그만큼 줄어든 것으로 느껴지며, 따라서 지연 시간이 줄어든 대역폭만큼 증가한다. 여기서 주의할 점은, 이 지연 시간 증가는 홈 게이트웨이의 큐를 통과하는 데 걸린 시간만을 고려한 것이라는 점이다. 최종적인 지연 시간은 패킷이 지나가는 모든 링크와 노드에서의 지연 시간을 함께 고려해야 하기 때문에 구체적인 네트워크 구성과 상태에 따라 크게 달라질 수 있다.

4.2 Queuing Discipline을 사용한 구현

우리는 개발한 홈 게이트웨이의 네트워크 서브시스템을 수정하여 개선된 구조의 출력 큐를 구현하고 적용하였다. 이제 그 구현 내용과 적용 결과를 설명한다. 우리는 리눅스의 queuing discipline기능을 이용하여 제안된 출력 큐를 구현한다. 이 기능은 사용자가 런타임에 시스템 콜을 통하여 입/출력 큐를 다양한 형태의 복잡한 큐로 구성할 수 있도록 하는 기능이다. 개선된 홈 게이트웨이 시스템의 성능을 측정한 결과 업로드와 다운로드를 동시에 진행시킬 때 업로드는 약 750 Kbps, 다운로드는 약 2 Mbps로 각각 링크의 최대 전송 속도에 근접하는 전송 속도를 달성하여 속도 간섭 문제가 해결됨을 확인했다.

출력 큐의 구현을 설명하기에 앞서, 구현에 사용된 queuing discipline기능에 대해 설명한다. Queuing discipline[9]은 리눅스의 네트워크 입/출력 큐의 구성을 런타임에 시스템 콜을 통하여 수정할 수 있도록 해 주는 기능이다. 사용자는 기본적으로 설정된 FIFO 큐를 대체하여 다른 policy를 구현하고 있는 큐들을 설치할 수 있고 이러한 큐들을 계층적으로 배치하여 더욱 복잡한 policy를 구현하는 큐를 구성할 수도 있다.

Queuing discipline에서 큐는 classless 큐와 classful 큐의 두 가지로 나뉜다. Classless 큐는 내부에 다른 큐들을 가질 수 없는 단일 큐로 FIFO, 우선순위 큐, RED 등이 있다. Classful 큐는 내부에 다른 classless나 classful 큐들을 가질 수 있는 큐로써, CBQ, HTB등이 있다. Classful 큐의 내부에는 들어온 패킷을 어떤 내부 큐로 전달할 지를 정하는 패킷 필터 객체들이 있으며, 반대로 패킷을 하나 가져와야 할 경우 어떤 큐에서 패킷을 가져올 지를 정하는 패킷 스케줄러 객체가 존재한다. 사용자는 패킷 필터가 가려 낼 패킷의 조건과, 조건을 만족하는 패킷일 경우 어떤 내부 큐로 패킷이 전달될지를 정한다. 마찬가지로 패킷 스케줄러를 설정하기 위해서는 classful 큐의 종류에 따라 각 내부 큐의 우선순위나, 선택될 비율을 설정한다. 그림 8은 queuing discipline에서 구성된 큐의 한 예를 보여주고 있다. 패킷이 패킷 필터의 조건을 만족하지 않을 경우 다음 패킷 필터의 조건을 검사하게 되며, 다음 패킷 필터가 존재하지 않을 경우 그 패킷은 default로 지정된 큐로 전달된다. 어떤 큐들은 큐에 패킷이 삽입되는 속도를 제한할 수 있는 기능을 가지고 있다. 예를 들어 HTB (Hierarchical Token Bucket)은 classful 큐의 한 종류인데 이 큐는 토큰 버킷을 사용하여 패킷의 삽입 속도를 특정 속도 이하로 제한할 수 있다.

속도 간섭 현상을 해결하기 위한 출력 큐는 queuing discipline을 이용하여 그림 9와 같이 구현된다. 우선 최상위에 classful 큐로 HTB가 설치되며, 이 큐의 내부에는 classless 큐인 FIFO 큐가 두 개 설치되고 크기를 100으로 설정한다. 이들 중 한 큐는 ACK 패킷을 위한 것이며, 다른 하나는 ACK 이 아닌 다른 모든 패킷들을 위한 것으로 후자의 큐가 default 큐가 된다. HTB의 filter 객체는 ACK 패킷을 검출하면 전자의 FIFO 큐에 패킷을 전달하도록 설정된다. 마지막으로 ACK 패킷을 위한 큐(전자)가 다른 큐(후자)보다 높은 우선순위를 갖도록 설정하여 ACK 패킷이 지연 시간 없이 처리되도록 한다.

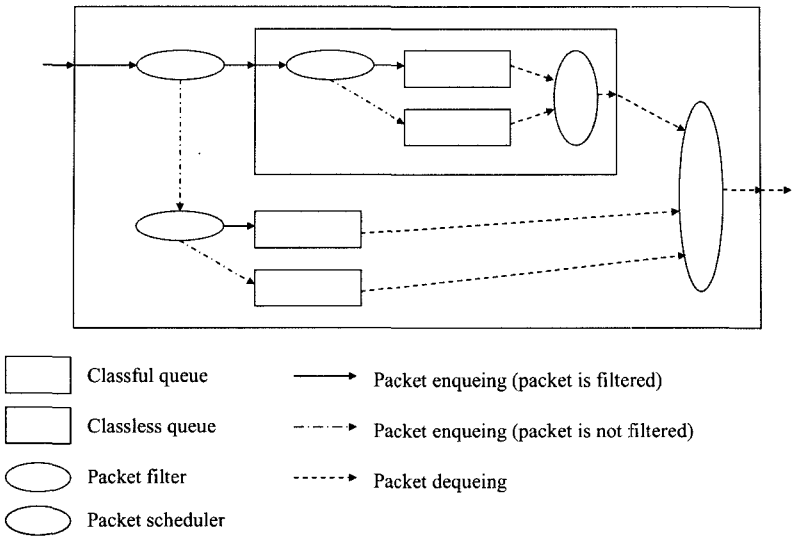

그림 8 Queuing discipline으로 구성된 큐의 예

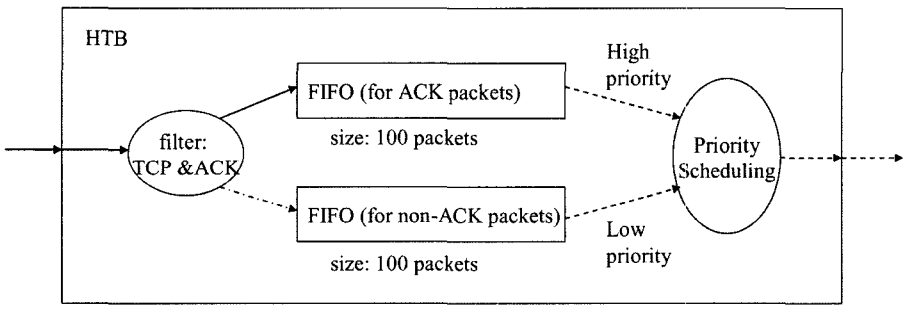

그림 9 Queuing discipline을 이용하여 홈 게이트웨이에 구현된 출력 큐의 모습. 범례는 그림 8과 동일하다.

4.3 실험 결과

이제 실험을 통해 홈 게이트웨이에 구현된 이 해결 방안이 잘 동작하는지 알아본다. 비교를 위해 출력 큐로 100의 크기를 FIFO 큐를 사용하는 경우와, ACK 패킷을 우선 처리하도록 개선한 큐를 사용하는 경우의 두 가지 경우에 대하여 속도 간섭 현상이 나타나는지 살펴본다.

그림 10은 이 실험의 결과를 보여주고 있다. 단일 FIFO 큐를 사용할 경우에는 업로드 속도가 약 300

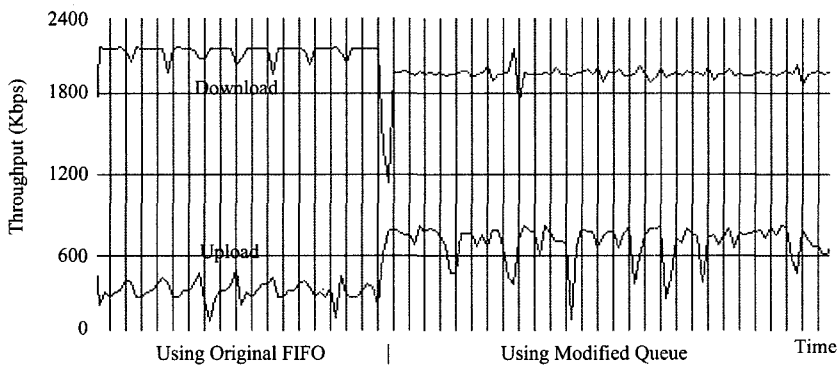

그림 10 기존의 FIFO 출력 큐와 개선된 출력 큐 사용에 따른 업로드/다운로드 속도의 변화. 다운로드 속도는 약 2 Mbps, 업로드 속도는 약 750 Kbps로 속도 간섭 현상이 해결됨

Kbps정도로 최대 속도인 800 Kbps보다 훨씬 느린, 속도 간섭 현상이 발생한다. 그러나 개선된 큐 구조를 사용할 경우 다운로드 속도는 2 Mbps, 업로드 속도는 750 Kbps로 링크의 최대 전송속도인 2.1 Mbps와 800 Kbps 에 매우 근접한 속도가 달성된다. 따라서 이 실험을 통해 ACK 패킷을 위한 별도의 큐를 두고 이 큐에 우선순위를 두어 패킷 스케줄러에 의해 처리하도록 한 개선된 큐 구조가 속도 간섭 문제를 효과적으로 해결함을 알 수 있다.

이 실험 결과에서 추가적으로 언급할 것은 다운로드, 업로드 속도가 링크의 최대 전속 속도에 약간 못 미쳤다는 것이다. 이들은 각각 4.8%, 6.3% 감소하였다. 이 감소는 각 링크로 전송되는 ACK 패킷 때문에 발생한 것이다. 이러한 성능 감소는 TCP 프로토콜을 쓰는 한 제거할 수 없는 오버헤드이다.

5. 결 론

본 논문에서 상용 홈 네트워크 게이트웨이를 개발하면서 비대칭 링크에서의 속도 간섭 문제를 발견하고 해결하는 과정을 소개하였다. 이러한 문제는 보통 receive livelock 문제로 생각되어 입력 큐에서의 패킷 경쟁이 그 원인이라고 분석된다. 그러나 우리는 실험을 통해 속도 간섭 문제가 receive livelock과는 그 원인이 다른 새로운 문제임을 밝혀내었다. 추가적인 분석과 실험을 통해 속도 간섭 문제의 원인은 TCP 프로토콜의 데이타 패킷이 ACK 패킷 때문에 출력 큐에서 유실되었기 때문임을 밝혀내었다. 이러한 분석을 바탕으로 우리는 ACK 패킷과 ACK 이 아닌 패킷을 위한 두 개의 큐를 두고 ACK 패킷을 우선적으로 처리하는 새로운 출력 큐 구조를 설계하고 홈 게이트웨이에 구현했다. 그 결과 속도 간섭 현상을 완전히 해결할 수 있었다.

이러한 연구 결과는 다음과 같이 확장될 수 있다. 첫째, 우리는 ADSL 보다 업로드 다운로드 대역폭의 비대칭성이 더욱 크고 긴 지연시간을 가지는 링크에서의 속도 간섭 현상을 해결하는 방법을 연구하고 있다. 이러한 링크의 대표적인 예로는 위성 링크가 있다. 이런 환경에서는 다운로드 트래픽을 위한 ACK 패킷만으로도 업링크의 모든 대역폭이 소비되어 업로드 트래픽을 위한 데이타 패킷이 전송될 여유 대역폭이 사라지는 문제점을 해결해야 한다. 둘째, 우리는 궁극적으로 이러한 연구 결과들을 종합하여 네트워크의 대역폭, 지연시간 등을 파라미터화 하여 비대칭 링크에서의 속도 간섭현상에 대한 일반적인 모델을 만들고 이에 대한 해결책을 제시하고자 한다.

참 조 문 헌

[1] Jeffrey C. Mogul, "Eliminating Receive Livelock in an Interrupt-Driven Kernel," *ACM Transaction on Computer Systems (TOCS)*, vol. 15, no. 3, pp. 217-252, Aug. 1995.

[2] T.V. Lakshman, U. Madhow, "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance," *Proceedings of INFOCOM.*, pp. 1199, Apr. 1997.

[3] L. Kalampoukas, A. Varma, "Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions," *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and Modeling of Computer Systems*, pp. 78-89, Jun. 1998.

[4] H. Balakrishnan et al., "The Effects of Asymmetry on TCP Performance," *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pp. 77-89, Sep. 1997.

[5] "uClinux: Embedded Linux/Microcontroller Project," *http://www.uclinux.org*

[6] "DSL Forum," *http://www.dslforum.org*

[7] Jacobson, V., "Congestion avoidance and control," *ACM SIGCOMM Computer Communications Review*, vol. 18, no. 4, pp. 314-329, Aug. 1998.

[8] Chu, H. H., Nahrstedt, K., "CPU service classes for multimedia applications," *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1999.

[9] Bert Hubert, "Linux Advanced Routing & Traffic Control HOWTO," *http://lartc.org/howto*

홍 성 수

1982년~1986년 서울대학교 컴퓨터공학과(B.S.). 1986년~1988년 서울대학교 컴퓨터공학과(M.S.). 1988년~1989년 한국전자통신연구소(연구원). 1989년~1994년 University of Maryland, Department of Computer Science(Ph.D.). 1994년~1995년 University of Maryland, Department of Computer Science(Faculty Research Associate). 1995년~1995년 Silicon Graphics Inc.(Member of Technical Staff). 1995년~1997년 서울대학교 전기공학부 전임강사. 1997년~2001년 서울대학교 전기공학부 조교수. 2001년~현재 서울대학교 전기컴퓨터공학부 부교수