논문 2004-41TC-7-8

고속 병렬 패킷 여과를 위한 효율적인 단일버퍼 관리 방안

(An Efficient Central Queue Management Algorithm for High-speed Parallel Packet Filtering)

임 강 빈*, 박 준 구**, 최 경 희***, 정 기 현****

(Kangbin Yim, Junku Park, Kyunghee Choi, and Gihyun Jung)

요 약

본 논문은 고속의 병렬 패킷 여과를 위한 다중프로세서 시스템이 가지는 단일 버퍼에서 단일 버퍼의 판독을 위한 다중프로세서 간의 경합을 중재하기 위한 효율적인 단일 버퍼 관리 방안을 제안하고 이를 실제의 다중 프로세서 시스템에 적용하여 실험함으로써 제안한 방안이 납득할 만한 성능을 제공함을 증명하였다. 병렬 패킷 여과시스템으로는 처리의 고속화를 위하여 패킷 여과규칙을 다중의 프로세서에 걸쳐 분산 처리하는 경우를 모델로 정하였다. 실제의 실험은 다중 프로세서를 가지는 네트워크 프로세서에서 이루어졌으며 100Mbps 의 통신망을 배경으로 하였다. 제안한 방안의 성능을 고찰하기 위하여 프로세서 수의 변화 및 여과 규칙의 처리 시간의 변화 등에 따르는 실제 패킷 전송률을 측정하였다.

Abstract

This paper proposes an efficient centralized single buffer management algorithm to arbitrate access contention among processors on the multi-processor system for high-speed packet filtering and proves that the algorithm provides reasonable performance by implementing it and applying it to a real multi-processor system. The multi-processor system for parallel packet filtering is modeled based on a network processor to distribute the packet filtering rules throughout the processors to speed up the filtering. In this paper we changed the number of processors and the processing time of the filtering rules as variables and measured the packet transfer rates to investigate the performance of the proposed algorithm.

Keywords: Central Queue, Parallel Processing, Packet, Packet Filtering Rule

I. 서 론

인터넷이 빠른 속도로 성장하면서 비전문가라도 인터넷을 용이하게 접근할 수 있을 정도로 보편화 되었다. 아울러 해킹 등 인터넷 범죄에 사용되는 가능성도 매우 높아지고 있다. 발표 자료에 의하면, 인터넷 해킹에 의한 범죄가 매년 4.5배씩 증가하는 것으로 보도되고 있으며, 피해 규모도 매우 빠른 속도로 증가하고 있다. 이러한 이유로 컴퓨터 네트워크뿐만 아니라 컴퓨터 내에 보관된 정보의 해킹을 방지하기 위한 보안 기술에 대한 관심이 매우 고조되면서 방화벽(Firewall)이나 침입탐지시스템(Intrusion Detection System) 등과 같은 보안 장비에 대한 연구도 많은 학자들에 의하여 진행되고 있다.

일반적으로 방화벽이나 침입탐지시스템은 개방된 인터넷으로부터 보호하고자 하는 LAN 으로 들어오는 모든 패킷에 대하여 그들이 해킹과 무관함을 평가한다. 특히 방화벽은 주로 외부에서 내부 망으로 들어오는 소위 ingress 패킷들의 적법성을 평가하며, 부적합하면 폐

* 정회원, 순천향대학교 정보보호학과
(Dept. of Information Security Engineering, Soonchunhyang University)
** 준회원, 디지털스트림테크놀로지
(Digital Stream Technology Inc.)
*** 아주대학교 정보 및 컴퓨터공학부
(School of Information and Computer Engineering, Ajou University)
**** 아주대학교 전자공학부
(School of Electronics Engineering, Ajou University)
※ 이 논문은 과기부 국가지정연구실사업의 지원에 의한 연구 결과임
접수일자: 2004년3월5일, 수정완료일: 2004년6월21일

기함은 물로, 보다 적극적으로 부적합한 패킷이 관여한 TCP 연결을 단절시키는 등의 적극적 대처를 하기도 한다. 이러한 일련의 과정에서 나타나는 패킷의 적법성 평가 작업에 패킷의 적법성 여부를 정의하는 패킷 여과 규칙(filtering rules)을 이용한다. 여과 규칙은 일반적으로 ACL(Access Control List)이라는 구조체로 표현되며, 조건(conditions)과 그 조건에 맞는 패킷의 처리를 정의하는 동작(action) 부분으로 구성된다. 패킷이 입력될 경우 방화벽은 그 패킷의 헤더 부분을 검출하여 자신이 보유한 ACL의 조건 부분을 비교한다. 이를 매칭(matching)이라 한다. 매칭되는 여과규칙을 찾은 경우 그 action에 따라 패킷을 내부 망으로 전달하거나 혹은 패킷을 폐기하는 두 가지 선택이 있을 수 있다.

방화벽에서의 패킷의 적법성 평가는 입력되는 모든 패킷에 대하여 매칭되는 여과규칙을 결정하기 위하여 자신이 보유한 모든 여과규칙에 대하여 그리고 각 여과규칙에 제시된 모든 조건에 대하여 매칭 여부를 판단하기 위한 비교작업을 수행한다. 따라서, 매칭을 처리하기 위하여는 매우 큰 작업 부하를 가지게 되어 방화벽은 매우 고속으로 동작할 것이 요구된다. 이에, 경우에 따라서는 그만을 위한 특수한 전용 하드웨어로 구성되는 경우도 있다. 통신망의 전송 속도의 발전에 비례하여 고성능 시스템에 대한 요구가 날로 증가하고 있으므로, 통신망의 발전 경향을 볼 때 전용 하드웨어에 의한 구성은 더욱 증가할 것으로 예상된다.

현재의 사설망에서 바라보는 최고 속도의 통신망으로 말할 수 있는 기가 대역 통신망의 경우 최대 1Mpps (Million Packets per Second)을 처리하여야 하므로[6] 라우터나 스위치, 방화벽과 같은 통신 장비에서 이러한 고속 처리의 요구를 충족시키기 위하여 다양한 방법이 제안되고 있다. 가장 대표적인 방법으로 라우터나 스위치, 방화벽 등의 장비에서 주어진 시스템의 성능에 대하여 전체 처리 능력에 큰 영향을 미칠 수 있는 여과규칙 검색과 같은 테이블 검색 방안의 성능 향상을 위한 연구가 제안되었다. 이진검색 등의 방법과 그로부터 변형된 고속 검색 방안 등이 이에 속한다[12]. 하지만, 이러한 방법들은 IP Address 등과 같이 특별한 규칙이 있고 패킷 내에서 검색하는 부분이 일정하게 정해진 경우에는 효과가 있으나, 정형화된 규칙이 없을 경우에는 원하는 필드를 모두 비교할 수밖에 없다. 특히 소프트웨어적인 방법만으로는, N개의 규칙을 비교할 경우 O(N)의 수행시간을 가지게 되어 기가급 고속 망에서는 적합하지 않은 것으로 판단된다.

이에, 많은 시스템들이 시스템의 기반이 되는 하드웨어의 고속화를 위하여 Multi-threaded Architecture를 사용하고, 시스템 버스와 별개로 분리된 Network Bus를 유지함으로써 시스템 내부의 데이터 전송에서의 병목현상을 해결하였다.[1][2] 또한 칩 수준에서의 다중 프로세서를 사용하는 방법이 제안 되었으며, 다중 프로세서 환경에서 상위 소프트웨어를 패킷 수준의 병렬성을 고려하여 구현하는 등[7], 성능 향상에 더 많은 이득을 볼 수 있는 방안들이 제안되었다. 또한, 이러한 하드웨어적 지원을 특화 시켜 통신망의 응용에 적합하도록 설계된 네트워크 프로세서도 등장하였다. 인텔의 IXP1200은 이러한 구조를 가진 전형적인 네트워크 프로세서이다. 다중 프로세서 네트워크 프로세서가 제안되면서 그들의 하드웨어 구조에 적합한 병렬 검색을 지원하기 위한 방안들이 제안되었다. 이러한 제안들 중에 방화벽을 위한 ACL에서각 요소간의 종속성을 해결 하여 순차적이 아닌 병렬 처리를 가능하도록 하는 방안이 있다[9]. 이 방법에 의하면 논리곱으로 표현되는 순차적인 검색 요소들을 풀어 논리합으로 변환하여 각 항목에 독립성을 제공함으로써 패킷 평가의 병렬처리가 가능하다.

병렬처리에 의한 패킷 적법성 평가를 위하여는 병렬처리를 위하여 정의되는 각 프로세서 또는 태스크들을 위하여 패킷을 공급해 주기 위한 버퍼가 사용된다. 다중의 프로세서 또는 다중 태스크 분산 환경에서 사용되는 버퍼 모델(Queue Model)로는 단일 버퍼 모델(Centralized Queuing Model)과 분산 버퍼 모델(Distributed Queuing Model)이 있으나 단일 버퍼 모델이 분산 버퍼 모델에 비하여 속도가 빠른 것으로 평가되었다[14]. 그러나, 단일 버퍼를 사용할 경우에는 다중 프로세서가 하나의 버퍼를 사용면서 버퍼 경합이 발생하며, 이의 해결이 병렬 알고리즘과 패킷 여과의 성능에 많은 영향을 주게 된다. 본 논문에서는, 인텔 IXP1200 과 같이 내부에 다중 프로세로를 가진 네트워크 프로세서 환경에서, 고속의 단일 버퍼를 사용하는 병렬 패킷 여과 기법에서 발생하는 경합을 해결하기 위한 효과적인 버퍼 관리 방안을 제안한다. 제안된 기법은 다중의 프로세서에서 실행되는 태스크들이 동시에 참조하는 단일 버퍼에 대한 경합 문제를 효율적으로 해결하고 있다. 제안 기법을 구현한 후 IXP1200 시뮬레이터에서 성능을 평가하였다.

본 논문의 구성은 다음과 같다. 제 II장에서는 패킷 여과 규칙의 병렬 처리와 그에 관련한 연구에 대하여 기술한다. 제 III장에서는 본 논문에서 제안하는 병렬 패킷 여과를 위한 단일 버퍼의 효율적인 관리 방안에

대하여 자세히 설명한다. 제 Ⅳ장에서는 제안하는 방안에 대한 실험과 그 결과에 대하여 논하며, 제 Ⅴ장에서 결론을 기술한다.

Ⅱ. 병렬 패킷 여과에 관한 관련 연구

패킷 여과란 들어오는 패킷의 내용을 검사하여 매치되는 규칙을 판단한 후, 규칙에 정의된 처리 방법에 따라 패킷을 폐기하거나 혹은 전달하는 과정을 말한다. 패킷 여과 규칙은 OSI 3계층에 해당하는 송신자 혹은 수신자 IP Address나 Type 등은 물론 OSI 4계층에 해당하는 송신자 포트와 목적지 포트 등을 기준으로 작성될 수 있다. 경우에 따라서는 그 이상의 계층에 해당하는 항목을 검사하기도 한다. 패킷 여과는 라우터나 스위치 등 네트워크 장비의 성능에 매우 결정적인 역할을 담당함은 물론, 방화벽이나 침입 탐지 시스템 등 망의 보안 시스템의 성능 향상을 위하여도 매우 중요하다. 이러한 이유로 이들 시스템들은 고성능의 패킷 여과 작업을 수행하기 위하여 패킷 여과 규칙을 효율적으로 작성하거나 유지하고, 효과적인 패킷 규칙 기법을 사용할 필요가 있다.

여과 규칙 테이블 검색의 속도, 즉 매칭 규칙 결정 속도를 개선하기 위한 연구는 다양한 방면에서 꾸준히 진행되어 왔다. 검색 알고리즘의 연구는 물론, 하드웨어를 이용한 방법도 꾸준히 연구되고 있다. 하드웨어를 기반으로 하는 방법은 대개 두 가지로 분류할 수 있다. 첫째는, 패킷 여과 ASIC[16]과 같은 패킷여과를 위한 특별한 하드웨어를 고안하여 사용하는 방법이다. 이 방법은 고속 처리가 가능하다는 장점은 가지고 있지만, 다양한 여과 규칙을 지원하기 어렵다는 단점과 보안 문제 등에 대한 유연성을 충족시키지 못하는 단점을 가진다. 둘째 방법으로 다수의 시스템을 사용한 다중 프로세서를 기반으로 한 병렬 패킷 여과[15]를 수행하는 방법이 있다. 이 방법은 각 프로세서에 규칙을 분산하고 패킷에 매치되는 규칙을 병렬적으로 결정하므로 프로세서 수에 따라 고속 처리가 가능하고, 각 시스템을 기존에 사용하던 장비로 재사용할 수 있다는 장점이 있다. 하지만, 패킷을 기존 이더넷 망을 통해 분산하기 위한 적절한 기법이 필요하며, 한 프로세서에 부하가 걸릴 경우 패킷손실의 위험이 있다는 단점을 가지고 있다. 이 문제를 해결하기 위해 단일 시스템에서 다중 프로세서를 지원하는 프로세서를 기반으로 한 효율적인 병렬 패킷 여과 방법들이 제안되고 있다. IXP1200은 가장 대표적 예라 하겠다.

그러나, 다중 프로세서를 내장한 네트워크 프로세서에서 병렬로 패킷을 처리하기 위하여는 다음과 같은 몇 가지 사항들이 고려되어야 한다. 우선, 병렬 패킷 여과를 위하여 패킷 여과 테이블의 각 여과 규칙이 서로 독립적이어야 한다. 다시 말하면, 하나의 여과 규칙이 다른 여과 규칙의 내용을 포함할 경우 패킷 여과 테이블의 검색 순서에 따라 불필요한 동작을 중복 수행함으로써 검색에 소요되는 처리 시간이 달라질 수 있으므로, 각 패킷 여과 규칙은 다른 패킷 여과 규칙에 대해서 완벽히 독립적이어야 한다. 그리고, 패킷 여과 규칙을 다중 프로세서 환경에서 수행할 경우, 각 패킷 여과 규칙이 서로 독립적이어야 병렬적으로 처리가 가능하다. 이러한 문제를 해결하기 위하여 IP Table 등에서 의존성을 제거하는 방법이 제안되었고,[9][17] 본 논문에서는 그러한 방법들이 사용되었다.

이런 방법을 통해 패킷 여과 규칙을 독립적으로 작성하더라도 처리되는 패킷 간의 의존성이 존재한다면, 병렬적으로 패킷 여과를 수행할 수 없다. 따라서 패킷 여과 규칙은 패킷들이 서로 의존적이지 않은 수준에서 작성되거나, 패킷이 의존적이라면 이를 고려하여 작성되어야 한다. 이러한 패킷 처리 과정에서 패킷 간에 의존성이 없다는 것을 패킷 레벨 병렬성[8][9]이라고 한다. 패킷 레벨 병렬성은 대개 OSI 2계층에서 발견할 수 있으며, 패킷 여과 규칙의 성격에 따라 그 이상의 계층에서도 가정할 수 있다. 본 논문에서는 모든 패킷은 패킷 레벨 병렬성을 유지하도록 패킷 여과 규칙을 작성하였다고 가정하였다.

다중 프로세서 환경에서 주어진 연산 능력에 대하여 최대의 여과 수율, 즉 처리한 패킷 수와 처리 대상 패킷 수와의 비율을 가능한 높게 하기 위하여는 패킷 여과 규칙들을 각 프로세서에 효과적으로 분산시켜야 한다. 이는 모든 프로세서의 수율을 가능한 크게 만드는 것이 전체적인 여과 수율을 크게 만들기 때문이다. 각 프로세서가 동일한 연산 능력을 가지고 있고 각 여과 규칙을 검사하는 데에 동일한 계산 시간이 필요하다면, 그리고 모든 패킷의 빈도가 같다면, 패킷 여과 규칙들이 각 프로세서에 균등하게 처리되도록 분산하는 것이 효과적이다. 하지만, 패킷 여과 규칙의 성격에 따라 검사 항목이 달라 처리 시간이 다르며, 패킷의 빈도도 다르므로 각 프로세서에 균등하게 패킷 여과 규칙을 분산하여 처리하는 것은 부하의 효과적 분산을 의미하지는 않는다.

Ⅲ. 병렬 패킷 여과를 위한 버퍼 관리 방안

본 장에서는 병렬 패킷 여과를 위하여 분산된 패킷 여과 규칙을 가지는 다중 프로세서 시스템에서, 고속의 패킷 여과를 위한 효과적인 단일 버퍼 관리 방안을 제안한다. 그림 1 은 시스템 내에서의 패킷의 흐름을 보여주고 있다. 통신망에 접속된 네트워크 인터페이스로부터 입력되는 패킷은 순차적으로 단일 버퍼에 채워지고, 각 프로세서는 단일 버퍼에 채워진 패킷을 읽어 여과 규칙과의 매치 여부를 판단한다. 이 때, 각 프로세서들은 자신에 할당된 여과 규칙들 중에서 어떤 규칙이 읽어 들인 패킷과 매치되는지를 판단한다. 각 프로세서는 자신이 가져간 패킷의 처리를 완료한 후에야 다음 패킷을 읽어 들인다.

또한, 앞의 제 Ⅱ장에서 논한 바와 같이 각 프로세서에 할당된 여과 규칙 간의 의존성은 없는 것으로 가정한다. 즉, 각 프로세서가 패킷을 처리하는 과정에서 여과 규칙 간의 서로의 의존성으로 인한 문제가 없다고 가정한다. 따라서, 버퍼에서 읽혀진 각 패킷은 여과 규칙에 의하여 이상 여부를 평가 받은 후 그 결과에 의하여 폐기되거나 출력단으로 전달된다. 그러나, 패킷의 빈도수를 고려하지 않고 각 프로세서가 모두 다른 여과 규칙만을 처리하는 경우에 빈도가 높은 패킷의 처리가 지연되어 전체적인 패킷 처리 수율이 저하될 수가 있다. 이에 본 논문에서는 최적의 여과 규칙 할당 정책은 논하지 않더라도 하나의 여과 규칙이 복수개의 프로세서에 할당되는 구조가 필요하다고 판단하여, 복수개의 프로세서가 동일한 여과 규칙을 검사할 수 있다고 가정한다.

프로세서에 여과 규칙을 할당하는 방법과 프로세서가 버퍼에 입력된 패킷을 인출하는 방법은 매우 밀접한 관계를 가지게 된다. 예를 들어, 모든 여과 규칙이 모든 프로세서에 동일하게 중복 할당된 경우에는 어떤 패킷이라 하더라도 일부 프로세서만이 인출하여 처리할 수 있다. 모든 여과 규칙이 모든 프로세서에 동일하게 중복 할당된 경우에는 각 프로세서에서 실행되는 여과 작업의 효율을 분산하는 알고리즘을 사용하여야 한다. Round-robin은 가장 쉬운 부하 분산 방법이라 하겠다.

그러나 프로세서가 일부 여과 규칙만을 처리하는 경우에는 패킷이 어떤 여과 규칙에 매치되는지를 미리 판별하지 않는 한 일부 프로세서만이 패킷을 가져가는 방법에는 무리가 있다. 더욱이 단일 버퍼에 도착하는 모든 패킷에 대하여 사전 판별하는 작업 자체에 많은 부하가 예상되기 때문에 패킷이 어떤 여과 규칙에 매치되는지를 프로세서에 할당하기 전에 판별하는 것은 무리이다. 따라서, 이 경우에는 모든 프로세서가 모든 패킷을 인출하는 것이 타당하며, 단일 버퍼로부터 패킷을 읽어 오는 과정에서는 다중 프로세서에 의한 패킷의 경합이 불가피 함을 의미한다. 따라서, 이러한 단일 버퍼 내에서의 패킷의 경합을 해결하기 위하여는 효율적인 성능의 패킷 경합 해결 방안이 요구된다. 단일 버퍼 내에서의 패킷 경합의 해결을 위한 방안에서 각 프로세서가 자신에게 할당된 여과 규칙을 패킷과 비교하는 과정에서 이 비교 시간을 최소화 하는 것은 여과 작업의 성능에 매우 중요한 요소이다. 이는 주어진 수의 패킷 여과 규칙 하에서, 전체 시스템이 처리하는 총 패킷의 수를 최소화함을 의미하며, 입력 버퍼 내의 모든 패킷을 처리하는 과정에서 각 프로세서마다 임의의 패킷에 대하여 중복 처리를 피할 수 있는 구조의 버퍼 관리 방안이 요구됨을 의미한다.

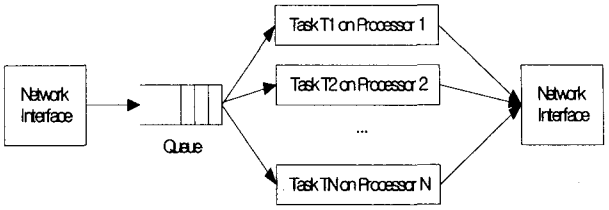

그림 1. 병렬 패킷 처리 모델
Fig. 1. A parallel packet processing model.

본 논문에서는 이러한 단일 버퍼 관리 방안을 제안한다. 단일 버퍼를 중재 관리하기 위하여 버퍼를 이루는 매 요소 즉, 패킷 관리를 위한 자료 구조체가 요구된다. 관리 구조체는 다음의 여러 가지 상황을 처리할 수 있어야 한다. 첫째, 패킷이 모든 프로세서에게 방송되는 효과를 얻기 위하여는 해당 패킷을 처리하였거나 처리하고 있는 프로세서의 수 즉, 경합 수위를 유지할 필요가 있다. 둘째, 임의의 프로세서가 특정 패킷을 처리하였는지를 프로세서 자신이 확인할 필요가 있으므로 이를 위하여 프로세서의 수만큼의 비트 수로 구성된 검색 완료 플래그가 요구되며 임의의 프로세서에게 할당된 특정 여과 규칙에 대하여 그 여과 규칙이 해당 패킷에 대하여 처리되었음을 알리는 규칙 처리 완료 플래그가 요구된다. 이 규칙 처리 완료 플래그의 수는 시스템 내의 모든 여과 규칙의 수와 같다. 마지막으로, 필요 없이 여과 규칙을 중복 검사하는 상황을 피하기 위하여는 임의의 여과 규칙에 대한 검사 결과, 패킷의 이상이 발견되었음을 나타내기 위한 사건 발견 플래그가 요구된다. 상

```
MyPacketPtr = 0;
while (1) {
    wait until (NbAccessed[MyPacketPtr] < N and PacketAccessed[MyPacketPtr] == False );

    Enter MUTEX;
        NbAccessed[MyPacketPtr] = NbAccessed[MyPacketPtr] + 1;
        PacketAccessed[MyPacketPtr] = TRUE;
        temp = RuleSetInQ[MyPacketPtr];
        RuleSetInQ[MyPacketPtr] = temp | RuleSetForTask;
        temp = RuleSetForTask & ~temp;
    Leave MUTEX;

        Get Packet from Queue.
        Perform packet matching for all rules Rj, one by one, corresponding to set bits
        in temp variable, until a rule that matches the packet is found
        or the RuleDetected is set.

        MyPacketPtr = MyPacketPtr + 1;
        if ( MyPacketPtr >= MaxQueueSize )
            MyPacketPtr = 0;
}
```

그림 2. 알고리즘
Fig. 2. Algorithm.

기와 같은 기능을 가진 버퍼를 위한 자료 구조체 이외에도 각 프로세서는 자신의 기능을 위하여 적절한 자료 구조체가 요구된다. 여과 규칙의 지역성(locality)에 따라 복수의 프로세서에 걸쳐 중복된 여과 규칙이 존재할 수 있으므로 각 프로세서는 자신에게 할당된 여과 규칙에 대한 정보 즉, 할당 규칙 상태를 유지하여야 한다. 이는 총 여과 규칙의 수만큼의 비트를 가지는 상태 집합으로 구현할 수 있다. 또한, 버퍼 내에서 현재 해당 프로세서가 처리하고 있는 패킷의 위치를 파악하기 위하여 패킷 포인터를 가져야 한다. 이 구조체의 상세한 내용은 다음과 같다.

그림 2는 제안하는 알고리즘이다. 알고리즘에서 사용한 각 변수들의 의미는 표 1에 기술된 바와 같다. Nb-Accessed, PacketAccessed, RuleSetInQ, RuleDetected 등은 각각 경합 수위, 검색 완료 플래그, 규칙 처리 완료 플래그, 사건 발견 플래그를 의미하며, RuleSetFor-Task, MyPacketPtr는 각 태스크 마다 정의되는 할당 규칙 플래그 및 패킷 포인터와 대응된다.

제안되는 알고리즘은 다음 과정을 통해 수행된다. 우선, 하나의 패킷이 패킷 버퍼에 도착하면, 모든 프로세서는 자신이 유지하는 패킷 포인터를 통하여 패킷 버퍼에 도착된 패킷에 접근하고, 해당 프로세서가 처리할 여과 규칙의 비트맵과 해당 프로세서가 패킷을 처리했다는 비트를 패킷에 표시한다. 패킷 구조체에 대한 처리가 끝나면 패킷 자체에 대한 여과 규칙 검사를 수행한다. 처리할 여과 규칙은 프로세서 자신이 유지하는 여과 규칙과 패킷에 이미 처리되어 있는 여과 규칙을 비교하여 결정한다. 여과 규칙 검사 도중에 이상 여부가 판단될 경우 즉, 그 프로세서가 가지고 있는 여과 규칙 중의 하나에 그 패킷이 여과되는 경우, 패킷 구조체에 이상 여부가 판단되었다고 표시한다. 다른 프로세서들은 이를 판독하여 도중에 이상 여부가 이미 인지되었

표 1. 사용된 변수
Table 1. Used variables.

변수명	의 미
NbAccessed	해당 패킷을 비교 및 처리한 태스크의 수를 나타낸다. 이 값이 전체 프로세서 수(N)이 되면 모든 태스크가 해당 패킷을 처리했다는 의미가 된다
PacketAccessed	해당 패킷을 처리한 태스크가 자신에 해당하는 비트 번호를 이 변수에 세트한다. 이 변수를 확인하여 자신이 해당 패킷을 처리했는지의 여부를 판단한다. (태스크 수 만큼의 비트 수)
RuleSetInQ	이 변수의 특정 비트가 세트 되는 것은 태스크가 자신에게 분산된 규칙을 처리하거나 처리했다는 것을 의미한다. 예를 들어, 3번째 규칙을 처리할 경우 3번째 비트를 세트하게 된다. 이 변수는 새로운 패킷을 넣을 경우 클리어 되고, 각 태스크가 처리할 경우 세트한다.(여과 규칙 수 만큼의 비트 수)
RuleDetected	패킷이 특정 규칙에 매치된 경우 세트 되며, 이 변수가 세트 되어 있으면 다른 태스크는 해당 규칙을 검색하지 않는다
RuleSetForTask	각 태스크에 유지되는 여과 규칙의 비트 문자열 변수이다. 예를 들어, 어느 태스크가 1번째 3번째 5번째 규칙을 유지할 경우 RuleSetForTask 변수는 101010b (0x2A) 의 값을 가지게 된다. 이 변수는 RuleSetInQ와 같이 사용된다. (태스크마다 가지는 변수, 규칙 수만큼의 비트 수를 가짐)
MyPacketPtr	각 태스크는 이 변수를 사용하여 버퍼에서 다음 패킷을 읽을 포인터를 유지한다. 이를 중재하기 위하여는 MUTEX(Mutual Exclusion) 등의 IPC 기구가 필수적으로 이용되어야 한다.

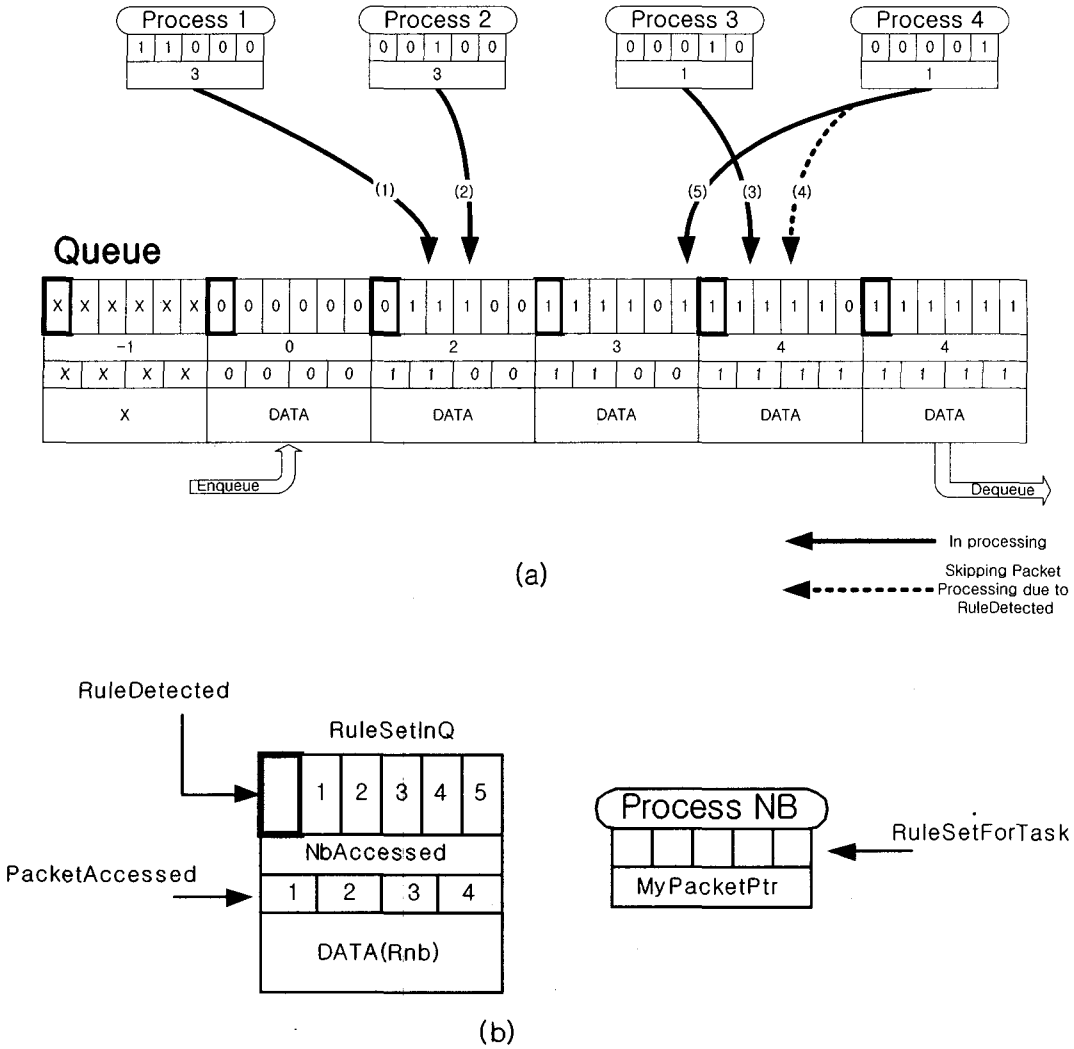

그림 3. 알고리즘 수행 과정, (a) 예제 (b) 구조체
Fig. 3. Process of the algorithm (a) example (b) structures.

다고 판단할 경우 더 이상 여과 규칙 검사를 수행할 필요가 없으므로, 작업을 종료하고 다음 패킷으로 진행한다. 이 때, 기본 동작이 전달이라고 정의 된 경우 이상이 있는 것으로 판단된 패킷은 폐기 되며 어떠한 여과 규칙에도 매치되지 않을 경우 전달 동작을 수행하게 된다. 동시에 해당 패킷은 패킷 버퍼에서 제거된다. 그림 3에서 이러한 과정을 설명하였다. 그림(b)의 좌측은 각 패킷에 대응하는 구조체이며 우측은 각 프로세서에 대응하는 구조체이다. 그림 (a)는 프로세서의 수가 4개이며 버퍼에 입력된 패킷이 5개라고 가정하여 그린 것이다. 이 때 패킷의 검색은 우측으로부터 좌측을 향하여 진행하는 것으로 그렸다. 그림에서 (1), (2)는 2개의 프로세서가 동시에 같은 패킷을 보고 있는 경우로서 해당 패킷의 NbAccessed가 2이며 프로세서 1이 가진 여과 규칙에 대한 값 11000b와 프로세서 2가 가진 여과 규칙에 대한 값 00100b에 의하여 RulesetInQ가 11100b가 됨을 볼 수 있다. 한편, PacketAccessed는 프로세서 1과 프로세서 2의 접근 사실을 유지함에 따라1100b가 됨을 볼 수 있다. (3), (4), (5)에서 (3)은Process 3이 패킷의 이상 여부를 판단하여 패킷에 이상이 있음을 발견하고 RuleDetected를 1로 하였으며 (4)는 Process 4가 이를 확인 하는 것을, (5)는 그 결과에 따라 이미 여과 결정이 끝난 패킷을 넘어 가는 것을 나타내었다. 그림 4에 이러한 과정을 흐름도로 나타내었다.

Ⅳ. 실험 및 평가

제안한 단일 버퍼 관리 방안의 성능 평가를 수행하였다. 제안한 방안의 구현은 네트워크 프로세서를 기반으로 하였다. 사용한 네트워크 프로세서 (Intel IXP1200)

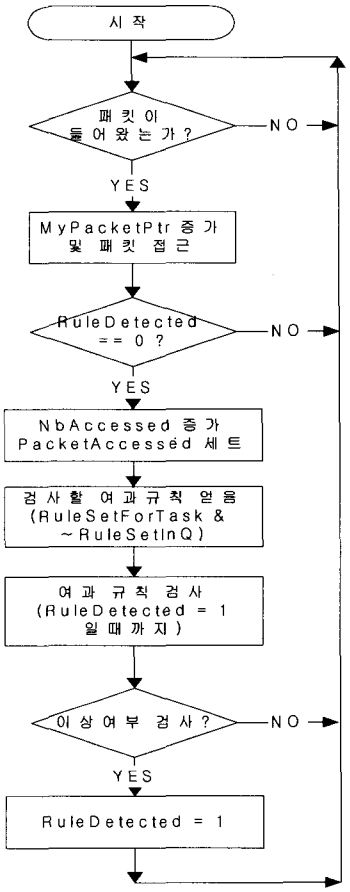

그림 4. 흐름도
Fig. 4. Flowchart.

는 그림 5와 같은 구조를 가지고 있다. 본 논문의 실험 환경으로서는 이미 언급된 네트워크 프로세서가 가지는 고속 처리를 위한 하드웨어 특성을 대부분 제공하는 시스템을 이용하였다. 사용한 네트워크 프로세서 IXP1200은 전체적인 관리를 위한 주 프로세서가 존재하며 이를 통하여 시스템의 메모리 버스가 제공된다. 또한 공유 메모리를 통하여 서로 연결된 여섯 개의 칩 레벨 RISC 멀티프로세서를 가지고 있으며 각각의 프로세서는 Multi-threaded Architecture로 이루어져 있다. 그리고 시스템의 메모리 버스와 구분되어 별개의 네트워크 버스를 제공하여 통신을 위한 인터페이스를 배려하고 있다.

패킷 수신과 단일 버퍼 및 패킷 여과 규칙의 구현은 상기의 RISC 프로세서를 이용한다. 각각의 RISC 프로세서를 위하여는 패킷 여과 규칙의 분산화에 의하여 주어진 여과 규칙들을 처리하기 위하여 동일한 구조의 태스크가 정의되어 수행되는데 실제 수행한 코드는 그림 2와 같은 알고리즘에 기반 하여 정의된다.

실험에 사용한 네트워크의 부하량은 특정한 도착 시간을 가지지 않고 항상 패킷이 도착한 상태라고 가정하

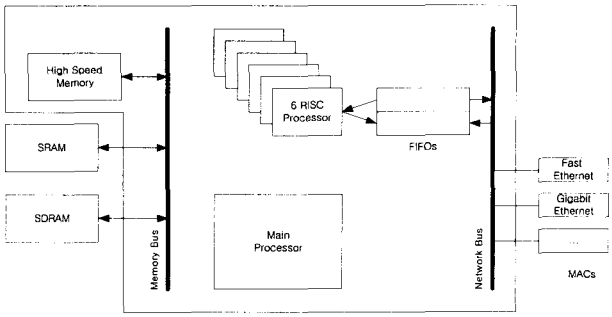

그림 5. IXP1200의 구조
Fig. 5. IXP1200 architecture.

여 생성된다. 이러한 부하량은 하위 네트워크 인터페이스에 무한한 패킷 버퍼가 존재하고 해당 네트워크의 속도가 무한대라고 가정함에서 기인하며, 이는 장비의 단위 시간 당 최대 패킷 처리량을 측정하는 상황에서의 패킷 고갈을 방지하기 위한 최소한의 조건이라 할 수 있다.

실험에서는 버퍼로 들어오는 패킷의 속도에 대한 고려와 함께 각 패킷의 내용에 대한 고려도 요구된다. 패킷 여과 규칙에 의한 적법성 평가에서 폐기되는 패킷이 많을수록 전달되는 패킷의 수는 증가하기 때문이다. 더구나, 패킷 여과 규칙에 일찍 매치될수록 더 많은 수의 남은 여과 규칙이 수행될 필요성이 없어지므로 그만큼 처리량은 증가하게 된다. 따라서 위와 같은 가정에 의하여 네트워크 부하는 여과 규칙을 고려하여 생성되어야 하므로 실험을 위하여 두 가지 경우로 네트워크 부하를 생성하였다. 첫 번째 네트워크 부하는 패킷 버퍼에 도착한 패킷들이 각 여과 규칙에 동일한 기회로 매치되도록 설정했다. 이런 경우 N개의 여과 규칙이 유지되고 N개의 패킷이 도착했을 경우, 각 여과 규칙은 평균적으로 1개의 패킷과 매치된다. 두 번째 네트워크 부하는 최악의 상황 도출을 위한 것으로서 도착한 패킷들이 어떤 여과 규칙에도 매치되지 않는 경우이며, 이러한 네트워크 부하를 사용할 경우 도착하는 패킷들은 모든 여과 규칙을 거쳐야 하므로 가장 큰 부하로 작용하며 상대적으로 많은 처리시간을 요한다.

실험을 위하여는 상기한 두 인자 이외에 처리하여야 하는 모든 여과 규칙의 수행 시간은 시스템의 총체적 성능에 영향을 주므로 매우 중요한 인자이다. 패킷을 실시간으로 손실 없이 처리해야 될 경우 패킷의 도착시간보다 처리시간이 작아야 하는데, 패킷의 크기가 작을수록 패킷간의 간격도 작아지고, 이에 따라 도착시간도 작아진다. 따라서 동일한 부하량에서 패킷의 크기가 작을수록 패킷을 처리하는 오버헤드가 더 크게 걸린다.

표 2. 여과규칙 수행시간
Table 2. Time for each filtering rule.

C(R0)	0.90us	C(R10)	0.90us
C(R1)	0.72us	C(R11)	0.72us
C(R2)	0.72us	C(R12)	0.72us
C(R3)	0.54us	C(R13)	0.54us
C(R4)	0.54us	C(R14)	0.54us
C(R5)	0.36us	C(R15)	0.36us
C(R6)	0.36us	C(R16)	0.36us
C(R7)	0.18us	C(R17)	0.18us
C(R8)	0.54us	C(R18)	0.54us
C(R9)	0.18us	C(R19)	0.18us

실험에서는 모든 여과 규칙의 수행 시간의 합이 100Mbps 데이터량에서 최소 패킷인 64바이트 크기의 패킷만 도착할 경우의 패킷 도착 시간 6.74us[6]을 기준으로 하여 실험을 수행하였다. 이를 기준으로 20개의 여과 규칙의 수행 시간이 표 2와 같이 분포하는 것으로 가정하였으며, 각 여과 규칙의 수행시간의 합은 6.74us의 1.5배가 되는 10.11us가 되도록 가정하였다. 즉, $\sum_{i=0}^{19} C(R_i) = 10.11\,us$ 으로서 여기에서 C(R_i)는 여과 규칙 R_i가 가지는 연산시간을 의미한다. 이와 같이 수행시간을 최악의 경우보다 더 크게 가정한 것은, 패킷 버퍼 관련 연산이나 패킷 전달과 같은 여과 규칙 이외의 수행 시간을 고려하기 위함이다.

실험에 사용되는 환경에서 각 프로세서는 동일한 컴퓨팅 파워를 가지기 때문에, 표 2에서 나타낸 여과 규칙 수행 시간을 기준으로 각 프로세서에 총 여과 규칙 수행 시간이 균일하도록 여과 규칙을 분산하였다. 이 때, 표에서 평균 수행 시간은 총 수행시간을 프로세서 수로 나누어 계산한 값이다. 표 3은 1개부터 5개의 프로세서를 사용할 경우 각 프로세서에 분산되어 할당되는 규칙과 이에 따른 각 프로세서의 평균 여과 규칙 수행시간을 나타낸 것이다. 실험의 환경으로 사용되는 IXP1200 시스템에서 지원하는 총 프로세서 수는 6개이지만, 1개 프로세서는 패킷 버퍼 연산 등의 여과 규칙 연산 이외의 수행에 사용된다. 그리고 여과 규칙 수행

표 3. 규칙의 배치
Table 3. Assignment of rules.

프로세서 수	규칙의 분산 배치	프로세서 당	
		총수행시간	평균수행시간
1	P1 : R0, R1, R2, ..., R19	10.11us	10.11us
2	P1 : R0, R1, R2, R7 R10, R11, R12, R17 P2: R3, R4, R5, R6, R8, R9 R13,R14,R15,R16,R18,R19	5.05us 5.05us	5.05us
3	P1 : R0, R2, R10, R12 P2 : R1,R3,R5,R11,R13,R15 P3 : R4, R6, R7, R8, R9 R14, R16, R17, R18, R19	3.25us 3.25us 3.61us	3.37us
4	P1 : R0, R6, R10, R16 P2 : R1, R3, R11, R13 P3 : R2, R4, R12, R14 P4 : R5, R7, R8, R9 R15, R17, R18, R19	2.53us 2.53us 2.53us 2.53us	2.53us
5	P1 : R0, R10 P2 : R1, R7, R11, R17 P3 : R2, R5, R12, R15 P4 : R3, R4, R13, R14 P5 : R6,R8,R9,R16,R18,R19	1.81us 1.81us 2.16us 2.16us 2.16us	2.02us

이외의 오버헤드는 시스템마다 다르고, 그 수치가 여과 규칙 수행보다 현저히 작으므로 무시할 수 있다.

그림 6은 앞서 설명한 두 가지 트래픽에 대하여 프로세서 수에 따른 평균 패킷 처리량을 나타내며, 그와 함께 실제 환경과의 비교를 위하여 100Mbps에서 최악의 경우 최대로 발생할 수 있는 초당 패킷량을 100Mbps 범례로 표시하였다. 그림에서 Worst Case는 앞서 설명한 바와 같이 최악의 경우 즉, 패킷이 어떤 여과 규칙에도 적용 되지 않고 모든 규칙을 통과해야 하는 경우를 의미하며 Random Case는 균일한 경우 즉, 패킷이 모든 여과 규칙에 동일한 확률로 적용 받는 경우를 의미한다. 예상한 바와 같이 프로세서 수가 증가할수록 처리량이 증가하였으며 실험 환경을 100Mbps로 가정하고 있지만 언급한 바와 같이 패킷은 무한대의 패킷 버퍼에 항상 도착해 있는 상태라고 가정하였으므로, Random Case에서와 같이 프로세서의 수가 4개 이상이 되면 100Mbps 이상의 패킷 처리율을 볼 수 있다. 이는 그 차이만큼 더 많은 입력에 대하여도 처리를 할 수 있음을 의미한다. 그러나, Worst Case의 경우 프로세서의 수가 5개가 되어야 패킷 처리율이 100Mbps에 접근하고 있다. 즉, Worst Case를 위하여는 최소한의 프로세서의 수가 6개가 되어야 100Mbps 이상을 처리할 수가 있다. 표 4에 프로세서의 수에 대한 성능 향상 정도를 단일

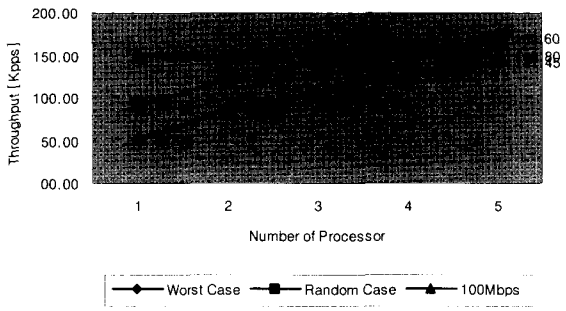

그림 6. 프로세서 수에 따른 패킷 처리량
Fig. 6. Packet throughput for different number of processors.

표 4. 프로세서 수에 따른 성능향상
Table 4. Gain for different number of processors.

프로세서 수	성능 향상
1	1
2	1.68
3	2.05
4	2.48
5	2.71

프로세서에 대비한 비율을 통해 나타내었다. 즉, 프로세서의 수가 2개인 경우에는 한 개인 경우에 비하여 1.67배의 성능 향상을 기대할 수 있으며 프로세서가 5개인 경우에는 2.71배의 성능 향상을 볼 수 있다.

패킷 처리량은 프로세서 수에 무조건 정비례하지 않고, 프로세서의 수가 증가할수록 패킷 처리량의 증가가 둔화되는 것을 볼 수 있는데 이러한 결과는 그림 2에서 보여지는 임계 영역에서 프로세서의 수가 증가함에 따라 그들이 사용하는 공유 변수 등을 위한 IPC 사용에 의하여 기다리는 시간이 증가하며, 더구나 공유 메모리 환경이므로 프로세서 수의 증가가 Memory Bus 사용에 의한 병목 현상을 유발했기 때문이다.

그림 7은 프로세서 수를 5개로 고정하고, 표 3에서와 같이 여과 규칙을 분산하여 표 2에서 정의한 여과 규칙의 수행 시간을 가변하면서 실험한 결과이다. 그림 6에서와 마찬가지로 100Mbps의 경우를 범례로 함께 나타내었다. Random Case와 Worst Case에 대한 그래프에서 100Mbps 그래프와의 교차점을 기준으로 하여 그 윗부분과 아래 부분으로 구분할 수 있는데, 윗부분이 100Mbps의 입력에 대하여 처리가 가능한 부분이다. 그러므로 Worst Case에서는 교차점의 x값인 8.98us가 100Mbps를 처리할 수 있기 위한 최대 처리 시간이며

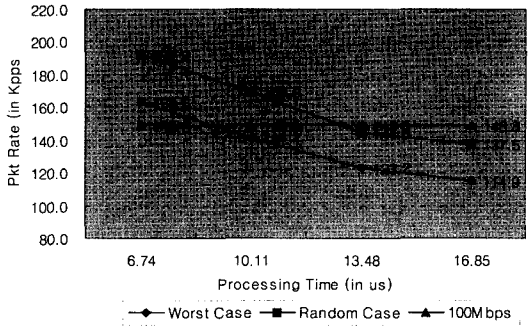

그림 7. 여과 규칙 수행 시간에 따른 처리량
Fig. 7. Packet throughput for rule processing time.

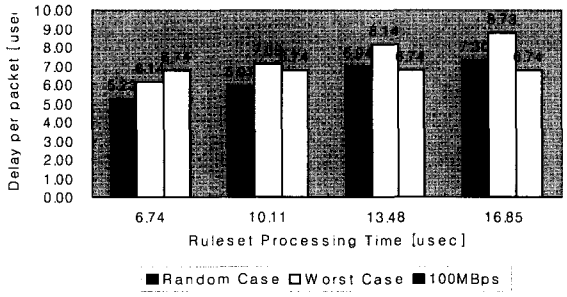

그림 8. 여과 규칙 수행 시간에 따른 패킷 당 수행 시간
Fig. 8. Processing time per packet.

마찬가지로 Random Case에서는 처리 시간이 12.9us 이하가 되어야 100Mbps를 손실 없이 처리할 수 있다. 이 때, 처리량의 변화폭이 수행 시간이 증가할수록 감소함을 알 수 있는데 이는 다음 수식과 같이 처리량이 수행 시간에 반비례하기 때문이다.

즉, $Throughput = K\frac{T_{IPG}}{T_{PROC}}$ 이며 이때, T_{IPG}는 주어진 대역폭에 대한 패킷간의 간격(inter-packet gap), T_{PROC}은 패킷 처리 시간, K는 주어진 대역폭에서의 최대 패킷 처리율을 의미한다.

각 상황에서의 패킷 처리를 위한 지연 시간을 위의 수식을 이용하여 계산하여 보면 그림 8에서 나타나는 바와 같다. 주어진 그림에서의 세로축이 나타내는 값은 단일 패킷을 처리하는 데 걸리는 평균 수행 시간을 의미하며 기준이 되는 100Mbps 대역에서의 최소 처리 시간과 각 상황에서의 처리 시간과의 차이를 비교해 볼 수 있다

그림 9에서는 프로세서의 수에 따른 패킷 버퍼 크기의 변화 양상을 보기 위한 것이다. 최대 패킷 버퍼 크기는 255개로 정하였는데 이는 패킷이 버퍼에 머무는 최대 시간을 40 단위시간으로 정함에 따라 얻어진 결과이다. 단위시간 1에서부터 패킷을 생성하다가 A의 시점에

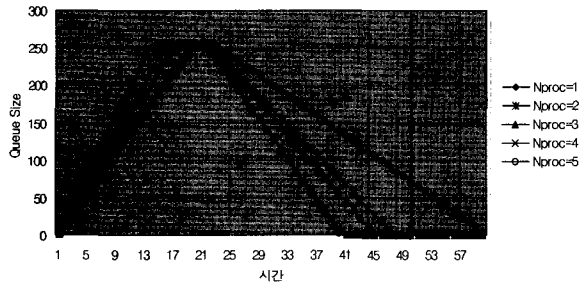

그림 9. 패킷 버퍼 크기 변화 (Queue Size = 255)
Fig. 9. Packet buffer size variation (Queue Size = 255).

서 패킷 생성을 중단했다. 실험에서, 네트워크 부하는 앞서 언급했던 바와 같이 하위 네트워크 인터페이스에 항상 도착해 있으므로, 패킷 버퍼는 지속적으로 증가한다. 이 때, 처리할 수 있는 양이 많을수록 그 증가 속도가 느리며 패킷 생성이 중단된 뒤에는 처리할 수 있는 양이 많을수록 감소속도가 빠름을 알 수 있다. 이에 따라, 그림 9는 주어진 최대 버퍼 지연에 따르는 최대 버퍼 크기를 선정하는 자료로 이용할 수가 있다.

V. 결 론

본 논문에서는 네트워크 프로세서와 같은 병렬 프로세서 시스템에서 통신망의 보안을 위한 패킷 여과 규칙을 병렬 처리하는 과정에서 단일 버퍼로의 프로세서 경합 문제를 해결하기 위한 버퍼 관리 방안을 제안하였다. 제안한 방안의 타당성 검증을 위하여 제안한 방안을 실제의 병렬 구조의 네트워크 프로세서에서 구현하여 실험하였으며 그 결과 프로세서 수의 증가에 따라 시스템의 총 패킷 처리량이 비례하여 증가함을 증명하였다.

실험에서 최악의 경우 프로세서 수가 5개 이상일 때 Fast Ethernet의 성능인 100Mbps의 성능을 만족하였고, 패킷의 검색 규칙에 대한 경우가 균일한 경우 프로세서 수가 4개 이상일 때 Fast Ethernet의 성능을 만족하였다. 이를 통하여 패킷 손실이 없이 패킷 여과를 수행하기 위한 프로세서의 수와 최대 수행 시간을 예측할 수 있었다. 그리고, 프로세서 수가 4개 이상일 때에는 IPC 및 메모리 버스 공유 등의 원인으로 성능 향상이 둔화되는 것을 볼 수 있었다.

현재 논문에서는 주어진 패킷 여과 규칙들이 다중의 프로세서 상에서 적절하게 배치되고 분산되었다고 가정하였으나, 실제로 패킷 여과 규칙들을 적절하게 배치 및 분산하는 것은 매우 어려운 문제이다, 그러나, 이러한 문제는 몇몇 연구에 의하여 조만간 해결될 것으로 보인다. 그리고, 네트워크 프로세서에서 지원하는 하드웨어 환경은 각 프로세서가 Multi-threaded Architec-ture이므로, 이는 단일 Context 환경에서의 설계와는 거리가 있다. 본 논문에서는 이를 무시하고 제안하였으나 좀 더 나은 성능을 위하여는 이를 고려한 설계가 연구되어야 할 것이다

참 고 문 헌

[1] Patrick Crowley, Marc E. Fiuczynski, Jean-Loup Baer, Brian N. Bershad, *Characterizing Proces-sor Architectures for Programmable Network Interfaces*, International Conference on Supercom-puting

[2] Patrick Crowley, Marc E. Fiuczynski, Jean-Loup Baer, *On the Performance of Multithreaded Ar-chitectures for Network Processors*, 2001.

[3] Tammo Spalink, Scott Karlin, Larry Peterson, *Evaluating Network Processors in IP Forwar-ding* ,Department of Computer Science, Prinston University, Technical Report, 2000.

[4] Prashanth Pappu, Tilman Wolf, *Scheduling Pro-cessing Resources in Programmable Routers*, Washinton University in St. Louis, 2001.

[5] T. Spalink, S. Karlin, L. Peterson, and Y. Gott-lieb, *Building a robust software-based router using network processors*, In Proc. Symposium on Operating Systems Principles (2001)

[6] Scott Karlin and Larry Peterson, *Maximum Packet Rates for Full-Duplex Ethernet*, Technical Report TR-645-02, February 14, 2002.

[7] Norman C. Hutchinson and Larry L. Peterson, *The x-kernel: An architecture for implementing network protocols*, IEEE Transactions on Soft-ware Engineering. 17(1):64-76, January 1991.

[8] James D. Salei, *Scheduling Parallel Networking on Shared-Memory Multiprocessors*, A Ph.D. Dissertation Proposal at Department of Computer Science, University of Massachusetts

[9] Takeshi Miei. Mitsuru Maruyama, Tsuyushi Ogura, Naohisa Takahashi, *Parallelization of IP-Packet Filter Rules*, IEEE 1997.

[10] *Intel IXP1200 Network Processor Family, Deve-lopment Tools user's Guide*, Intel, December, 2001.

[11] *DPF: Fast, Flexible Message Demultiplexing using Dynamic Code Generation*

[12] Venkatachary Srinivasan and George Varghese, *Faster IP Lookups using Controlled Prefix Ex*

-*pansion*, Measurement and Modeling of Com -puter Systems, p1-10, 1998.
[13] *Intel IXP1200 Network Processor Family, Hard -ware Reference Manual*, Intel Corp. December, 2001.
[14] Qiwei Huang, Errol Lloyd, *Analysis of Queuing Delay in Multimedia Gateway Call Routing*, CS REA, IC 2002.
[15] Carsten Benecke, *A Parallel Packet Screen for High Speed Networks*, University of Hamburg, Proceedings of the 15th Annual Computer Secu -rity Applications Conference 1999.
[16] J. Wu and M. Singhal. *Design of A High -Performance ATM Firewall*. In Proceedings: 5th ACM Conference on Computer and Commu -nications Security, San Francisco, California, November 4-5 1998.
[17] Thomas Y. C. Woo, *A Modular Approach to Packet Classification: Algorithms and Results*, INFOCOM, p1213-1222, 2000.

저 자 소 개

임 강 빈(정회원)
1992년 아주대학교 전자공학과 학사 졸업
1994년 아주대학교 전자공학과 석사 졸업
2001년 아주대학교 전자공학과 박사 졸업
현재 순천향대학교 정보보호학과 교수
<주관심분야: 시스템 보안, 실시간 운영체제, 임베디드 시스템, 멀티미디어 시스템 등>

최 경 희(정회원)
1976년 서울대학교 사범대학 수학교육과 학사 졸업
1979년 프랑스 그랑데꼴 ENSEIHT, 정보공학 및 응용수학 석사 졸업
1982년 프랑스 Univ. of Paul Sabatier 박사 졸업
현재 아주대학교 정보 및 컴퓨터 공학부 교수
<주관심분야 : 운영체제, 분산처리, 실시간 시스템 등>

박 준 구(준회원)
2002년 아주대학교 전자공학과 학사 졸업
2004년 아주대학교 전자공학과 석사 졸업
현재 디지털스트림테크놀로지 연구원
<주관심분야: 실시간 운영체제, 임베디드 시스템, 네트워크 시스템 등>

정 기 현(정회원)
1984년 서강대학교 전자공학과 학사 졸업
1988년 Univ. of Illinois, EECS 석사 졸업
1990년 Univ. of Purdue, 전기전자 공학부 박사 졸업
현재 아주대학교 전자공학부 교수
<주관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어 및 실시간 시스템 등>