패킷처리 응용을 위한 Click Modular Router의 최적화 방안*

김혜진°, 이재국, 김형식
충남대학교 대학원 컴퓨터공학과
angella@csalab.cnu.ac.kr, {empire, hskim}@cs.cnu.ac.kr

Optimization Methods of the Click Modular Router for Packet Processing Application

Hye-Jin Kim°, Jae-Kook Lee, Hyong-Shik Kim
Dept. of Computer Engineering, Chungnam National University

요 약

초고속 네트워크 인프라가 일반화되면서 네트워크간 트래픽 교환을 위한 라우터의 중요성이 커지고 있다. Click Modular Router는 소프트웨어적인 방법으로 라우터를 구현하기 위한 것으로, 다양한 기능을 접목시킬 수 있다는 점에서 활용 가치가 크다. 본 논문에서는 Click Modular Router를 활용할 때의 소프트웨어 오버헤드를 감소시키기 위한 네가지 최적화 방안을 제시하고, 성능 개선 정도를 보인다.

1. 서론

Click Modular Router(이하 Click)는 MIT LCS's Parallel and Distributed Operating System group과 Mazu Networks, ICSI Center, UCLA가 공동으로 개발한 모듈화된 소프트웨어 라우터로, 엘리먼트(element)라고 불리는 간단한 패킷처리 모듈들의 조합으로 구성되며, 'push connection' 또는 'pull connection'을 통하여 엘리먼트 사이의 패킷 전송이 이루어진다. 사용자는 목적에 부합하는 엘리먼트들을 선택하고 조합함으로써 다양한 종류의 네트워크 기능을 쉽게 구성할 수 있다[1].

Click은 소프트웨어 방식으로 패킷처리를 수행하며, 이와 같은 처리는 오버헤드를 발생시킨다. 본 논문에서는 성능 개선을 위하여 4 가지 최적화 방안을 제시하고, 성능 개선 정도를 확인한다.

2. Click Modular Router를 이용한 패킷 처리 구조

그림 1은 네트워크 A와 네트워크 B 사이에서 패킷을 포워딩하는 간단한 패킷처리 흐름도를 나타낸다. 네트워크 장치로부터 패킷을 받아 ARP 요청 패킷과 ARP 응답 패킷 그리고 IP 패킷으로 구분하여 처리하며, 라우팅 엘리먼트을 통하여 IP 패킷을 포워딩한다.

그림 1에서 A1과 B1은 네트워크 장치로부터 패킷을 받는 역할을, A2와 B2는 패킷의 이더넷(Ethernet)헤더를 제거한 다음 IP 주소를 얻는 역할을, A3와 B3는 큐를 통과한 패킷을 네트워크 장치로 보내는 역할을 한다.

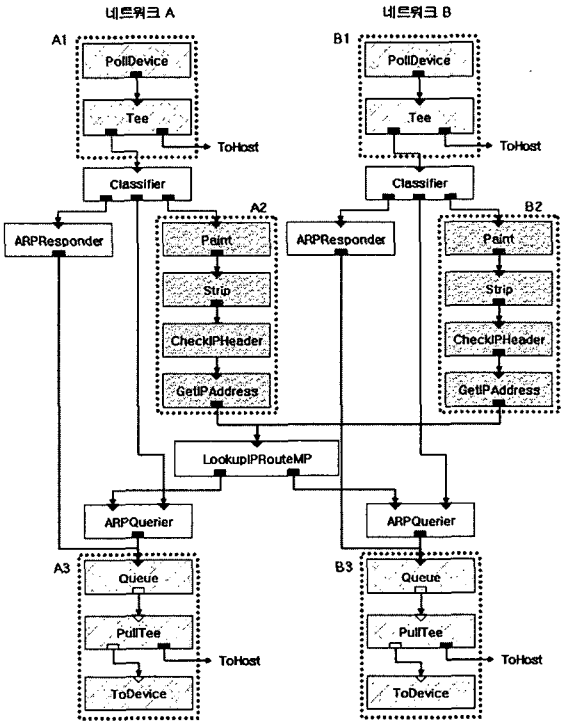

그림 1. 패킷처리 흐름도

3. 최적화 방안

본 절에서는 간단한 패킷처리 응용에 최적화를 위한 방안들을 적용한다. 최적화는 크게 엘리먼트를 대체하는 방법(방안 1)과 엘리먼트를 조합하는 방법(방안 2), 그리고

*본 연구는 "대학 IT 연구센터 육성지원사업"의 지원을 받아 수행한 연구 결과임.

Click에서 제공하는 툴을 사용하여 엘리먼트 내부 코드를 줄이는 방법(방안 3, 4)으로 나누어 진다.

[방안 1] 폴링(polling) 방식 채택

초기의 Click 시스템은 리눅스의 인터럽트 구조와 장치 처리 방식을 공유했다. 인터럽트 처리는 받는 패킷의 수가 증가할수록 다른 모든 시스템 작업이 실행되지 않는 상태를 초래하며, 전체적인 패킷 처리량을 오히려 감소시킨다. 이에 대한 대안으로, Click은 폴링 방식을 도입함으로써 인터럽트 처리의 오버헤드를 감소시키므로 시스템의 성능을 향상시킨다. 인터럽트 방식을 사용하는 'FromDevice' 엘리먼트를 대신하여 폴링 방식을 사용하는 'PollDevice' 엘리먼트를 이용한다.

PollDevice는 네트워크 장치를 폴링 모드로 변경한 다음 작업 큐에서 패킷을 기다린다. 엘리먼트가 활성화 되면 새로운 패킷이 들어오는지 감시하고, 발견되면 패킷 처리 구조를 통과하도록 한다. 폴링 방식은 인터럽트 구조의 오버헤드를 제거함으로써 패킷을 처리하는데 요구되는 시간을 줄일 수 있다.

[방안 2] 엘리먼트 개수 최소화

복합 엘리먼트(compound element)는 사용자로 하여금 내부 코드를 작성하지 않고서도 그들만의 엘리먼트를 정의할 수 있도록 한다. 이는 Click 언어의 추상화 메커니즘으로, 구성에 있어 유연한 매크로를 제공한다. 사용자는 기존의 엘리먼트들을 선택적으로 조합하여 조금 더 복잡한 처리를 할 수 있도록 별도의 입력 포트와 출력 포트를 갖는 하나의 엘리먼트를 생성할 수 있다. 패킷의 입장에서 보면 여러 개의 작은 엘리먼트들을 통과하는 것에서 하나의 엘리먼트를 통과하게 된다. 이와 같이 전체적인 구성에서 엘리먼트의 개수를 줄임으로써 패킷의 통과 경로를 단순화하고 처리 시간을 단축할 수 있다.

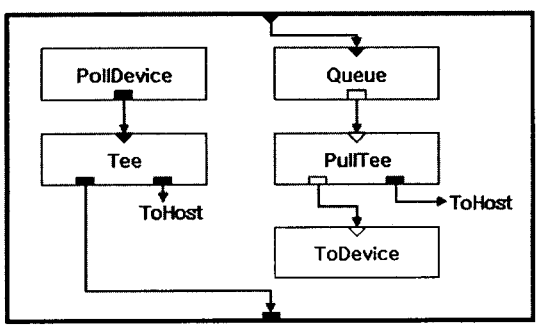

그림 2. GatewayDevice

그림 1의 패킷처리 흐름도에서 A1, A3와 B1, B3 부분을 조합하여 패킷을 네트워크 장치로부터 보내고 받는 역할만 전담하도록 그림 2와 같이 복합 엘리먼트인 GatewayDevice로 구성할 수 있다.

그림 3은 그림 1의 A2와 B2 부분을 복합 엘리먼트인 ExtractIPAddress로 구성한 것이다.

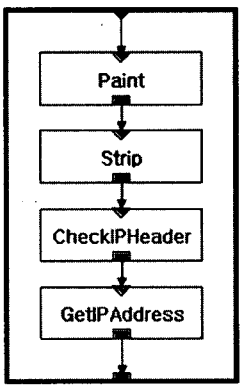

그림 3. ExtractIPAddress

[방안 3] 가상 함수 제거

엘리먼트들의 조합으로 이루어진 패킷처리 구성에서 엘리먼트간의 패킷 전송은 엘리먼트가 하는 일과는 무관하게 C++ 코드로 만들어진 'push'와 'pull'이라는 가상 함수를 통하여 이루어진다.

가상 함수 호출은 함수 테이블 포인터를 통하여 동적으로 읽어 실행되며, 잘못된 가상 함수 호출은 12 사이클의 지연을 수반한다[2].

```
Element *next = output(0).element;
// 가상 함수 테이블을 통한 호출
Next->push(output(0).port(),packet_ptr);
```

(a) 최적화 전

```
Classifier *next = (Classifier *)output(0).element();
// 컴파일 시간에 지정되는 호출
Next->Classifier::push(0,packet_ptr);
```

(b) 최적화 후

그림 4. 가상 함수 제거

그림 4는 click-devirtualize를 이용하여 가상 함수를 제거하기 전과 후의 코드 변화를 나타낸다. 패킷처리 구조가 구성되면, 각 엘리먼트의 위치 관계를 알 수 있으므로 (a)와 같은 패킷 전송 가상 함수 호출은 (b)와 같이 직접 연결된 엘리먼트의 함수 호출로 교체할 수 있다.

[방안 4] 결정트리 최소화

패킷 분류 규칙이나 패킷 필터링 규칙을 갖는 'Classifier', 'IPClassifier'와 'IPFilter' 같은 Click의 엘리먼트들은 실행시간에 규칙을 번역하여 결정트리(decision tree)로 파싱한다.

결정트리의 내부 노드들은 패킷 데이터의 한 워드(word)와 상수값 한 워드를 비교한다. 결정트리를 순회하면서 값을 비교하는 작업에 대한 오버헤드를 줄이기 위하여, click-fastclassifier는 트리 구조에서 상수를 인라인 처리하고 명시적으로 분기를 생성함으로써 규칙의 불필요한 서술 부분을 제거한다. 이는 실제 결정트리의 크기를 최소 2/3 이상 줄이는 효과를 보이는 것으로 알려져 있다[2].

또한, click-fastclassifier는 인접한 Classifier 엘리먼트들을 결합하여 하나의 빠른 패킷 분류 기능을 하도록 한다. 이는 패킷이 거쳐야 할 엘리먼트의 수를 감소하는 것으로 복합 엘리먼트와 동일한 효과를 낸다.

4. 성능분석

본 장에서는 3장에서 제시한 최적화 기법들을 적용하여 성능 개선 정도를 측정하였다.

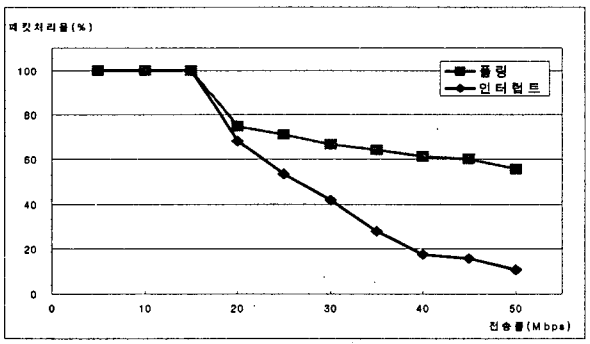

그림 5. 폴링 성능 분석

그림 5는 방안 1에 대한 성능 개선 정도를 보인다. 20Mbps 이상으로 많은 패킷을 전송할 때 방안 1에서와 같이 폴링을 채택하면 인터럽트를 사용할 때에 비하여 패킷 처리율을 크게 개선할 수 있음을 알 수 있다.

그림 6은 방안 1을 채택하고 방안 2, 방안 3, 방안 4를 각각 추가로 적용한 경우와 모든 최적화 방안을 통합하여 적용한 경우의 패킷 처리율을 나타낸다.

최적화 기법을 각각 적용한 경우에도 폴링 기법만을 적용한 경우에 비하여 패킷 처리율이 향상됨을 볼 수 있다.

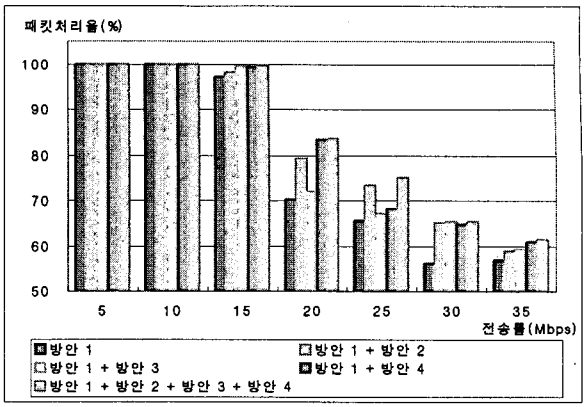

그림 6. 최적화의 성능 비교

5Mbps와 10Mbps의 전송률에서는 최적화 방안들의 패킷 처리율에 차이를 보이지 않는데, 이것은 전송해야 할 패킷의 수가 많지 않아서 최적화 여부와 관계없이 처리할 수 있기 때문이다. 그러나 15Mbps 이상의 전송률에서는 각 최적화 방안이 패킷 처리율을 향상시킴을 확인할 수 있다. 20Mbps의 전송률을 예로 들면, 모든 최적화 방안을 적용할 때 패킷 처리율이 10% 이상 증가하였다. 방안 1을 제외한 세 가지 방안 중에는 결정 트리 최소화(방안 4)를 적용한 경우가 상대적으로 큰 성능 개선을 나타낸 반면, 가상 함수 제거(방안 3)를 적용한 경우는 상대적으로 적은 성능 개선을 보인다.

5. 결론

본 논문에서는 패킷처리가 가능한 Click에서의 성능 개선을 위한 최적화 기법을 제시하였다. 그리고 각 최적화 기법들이 성능 개선에 영향을 미치는 정도를 알아보기 위해 간단한 시험을 하였다. 각각의 최적화 기법들은 성능 개선 비율에는 다소 차이를 보이지만 최적화 기법을 적용하기 전보다 성능이 개선되는 것을 확인할 수 있다. 최적화 방안을 모두 적용할 경우, 평균 15% 이상의 패킷 처리율을 기대할 수 있다.

참고문헌

[1] Click Modular Router(http://pdos.csail.mit.edu/click/)

[2] Eddie Kohler. "The click modular router", MIT, November 2000.