NET : Linux에서의 네트워크 에뮬레이터 구현

변상돈[0] 안종석
동 국 대 학 교 컴 퓨 터 공 학 과
{sdbyun,jahn}@dgu.ac.kr

NET : Network EmulaTor in Linux

Sang-Don Byun[0] Jong-Suk Ahn
Dept. of Computer Engineering, Dongguk University

요 약

물리적인 네트워크는 특정한 환경으로 설치된 후에는 고정적인 환경에서만 동작하므로 다양한 네트워크 환경을 요구하는 실험은 지원할 수 없는 문제를 갖고있다. 일례로 실험실 내부에서 WAN 환경의 네트워크를 기반으로 실험을 하고자 할 때 현실적으로 WAN 네트워크를 구현하는 것은 불가능하다. 따라서 하나의 실험실용 네트워크를 이용하여 다양한 형태의 네트워크를 에뮬레이션 할 수 있도록 지원할 수 있는 도구가 필요하게 된다. 본 논문에서는 Linux 시스템을 기반으로 하나의 네트워크에서 사용자가 요구하는 환경 설정 값으로 다양한 네트워크 토폴로지를 에뮬레이션 할 수 있는 네트워크 에뮬레이터를 설계, 구현하였다. 이 에뮬레이터는 사용자로부터 대역폭, 큐 지연시간 그리고 패킷 손실률 등을 입력 받아 능동적으로 네트워크 환경을 설정할 수 있다.

1. 서 론

다양한 형태의 네트워크 토폴로지에서 실험하기 위해서는 실제로 모든 종류의 토폴로지를 구성하여 실험하는 것이 바람직하지만 이는 현실적으로 불가능하다. 따라서 하나의 물리적인 네트워크에서 상황에 따라 여러 가지 형태의 물리적인 네트워크를 흉내낼 수 있도록 하는 기술이 필요하다. 일례로, QoS와 관련하여 다양한 서비스 모델을 실험하기 위해서는 직접 망을 구성하여 실험하여야 한다. 그러나 모든 형태의 망을 실제로 구성한다는 것은 불가능하기 때문에 일반적으로 네트워크 모델을 이용하여 시뮬레이션 실험을 한다. 그러나 정확한 네트워크 모델을 만드는 것은 매우 어렵거나 불가능하므로 시뮬레이션을 통해서는 정확한 결과를 도출할 수 없다. 또한 내부에서 실험을 하기 위해서 다양한 물리적 네트워크를 구성하는 것은 불가능하며 비효율적이다. 따라서 하나의 물리적 네트워크를 통해 여러 네트워크 토폴로지를 에뮬레이션 할 수 있는 도구가 필요하다. 이렇게 구성된 시험망은 기존의 물리적 네트워크에서 측정할 수 없는 네트워크의 성능과 문제점 그리고 그 원인을 쉽게 규명할 수 있는 장점이 있다.

에뮬레이터를 응용계층 차원에서 개발한 기존의 연구로는 두 가지를 들 수 있다. 첫째로, 적용 가능한 어플리케이션 개발을 위한 에뮬레이터[1]이다. 이 에뮬레이터는 응용계층에서 구현되었기 때문에 원거리 네트워크상에서의 프로세스 할당 지연시간은 무시되었다. 하지만 근거리에서는 전체 전송시간에 큰 영향을 준다는 문제점이 있다. 따라서 어플리케이션 레벨에서 수행하는 에뮬레이션은 적합하지 않다.

둘째로, WAN에서의 에뮬레이션 툴인 Delayline[2]이 있다. 이 에뮬레이터는 패킷 처리시간이 전체 에뮬레이션 수행 시간에 큰 영향이 없는 것으로 가정하고 있기 때문에 근거리 네트워크에서는 첫번째 툴과 동일한 문제점을 갖게 된다.

이 논문에서는 사용자가 하나의 물리적인 네트워크의 토폴로지와 환경을 설정해 주면 그 네트워크를 에뮬레이션 할 수 있는 툴인 NET를 설계하고 구현하였다. 기존 연구들에서는 일단 네트워크가 설정되고 나면 사용자가 그 네트워크의 환경을 다시 설정해 줄 수 없다는 단점이 있다. 이러한 문제점을 해결하기위해 NET는 커널에서 구현되었으며 사용자가 동적으로 자유롭게 환경설정을 변경시킬 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 NET의 구조와 디자인에 대하여, 3절에서는 구현한 NET에 관한 실험을 설명하고 마지막으로 4절에서는 실험결과의 요약과 향후 과제에 대하여 기술한다.

2. NET의 구조와 디자인

♦이 연구는 '99 대학기초연구사업의 지원을 받아 이루어졌습니다.

리눅스에서는 네트워크 트래픽 대역폭을 조절할 수 있는 shaper 기능이 지원되고 있다. 사용자가 이 프로그램을 모듈 형태로 커널에 적재하고, 초기 환경 설정시에 해당 디바이스의 대역폭을 지정해 준다. 그러면 shaper는 대역폭에 맞추어 해당 디바이스로 나가는 트래픽 대역폭을 조절하여 준다. 그러나 대역폭 외에는 다른 사항을 사용자가 지정해 줄 수 없다.

기존의 연구에서는 실제 네트워크를 통한 에뮬레이션이 아니라 단지 시뮬레이션을 하는 정도로만 시도되었다. 또한 사용자가 원하는 대로 네트워크의 설정 값을 완벽하게 지정해 주지 못한다. 따라서 본 논문에서는 NET를 커널 자체에 해당 알고리즘과 NET의 데이터 구조를 삽입하여 커널 수준에서 트래픽의 흐름을 조절할 수 있도록 한다. 또한 고정된 데이터 형태로 에뮬레이션을 수행하는 것이 아니고 사용자가 원하는 설정대로 여러 형태의 네트워크를 에뮬레이션 할 수 있도록 한다.

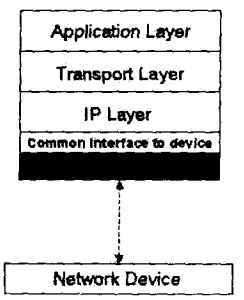

그림 1 NET를 포함한 TCP/IP 스택

<그림 1>은 NET를 포함한 TCP/IP 스택을 보여준다. 그림에서 보는 바와 같이 NET는 응용계층과 전송계층을 차례로 거친 패킷이 IP 계층을 거쳐서 네트워크 디바이스로 나가기 이전 부분에 삽입된다. 사용자는 대역폭, 큐 지연시간, 손실률 등의 설정값을 지정해 줄 수 있다. NET는 이 값에 의해 트래픽을 제어해 준다. IP 계층으로 들어온 패킷은 NET에 의해 timestamp가 생성되고 네트워크 디바이스 부분에서는 현재 시간과 해당 패킷의 timestamp를 비교하여 바로 전송할 것인지 아니면 지연을 시켜서 전송할 것인지를 결정한다.

2.1. 소켓버퍼와 커널의 수정

커널차원에서 NET를 구현하기 위해서는 해당 패킷이 나가게 될 timestamp를 저장하기 위한 공간을 소켓버퍼에 마련한다. 이 공간에는 해당 패킷이 소켓 버퍼에 들어온 시간에다 대역폭, 큐 지연시간, 그리고 손실률을 계산하여 산출된 지연시간을 더하여 전송될 시간을 기록한다. 디바이스에서는 패킷을 전송하기 전에 timestamp를 확인하여 현재시간과 비교한 후 전송여부를 결정한다.

2.2. 대역폭, 큐 지연시간 그리고 손실률

NET에서 사용하고 있는 모든 시간적인 계산은 기본적으로 Linux 커널에서 지원하는 지피(jiffie)값으로 한다. 이 시스템 클럭인 지피는 1 초당 100 지피로 되어있고, unsigned long 형으로 선언되어 있어서 계속적으로 카운트 했을때 1년 4개월 정도를 카운트 할 수 있다.

사용자가 대역폭을 설정하면 NET는 1 지피당 전송할 수 있는 비트를 계산하고 패킷이 디바이스로 나가기 전에 해당 패킷의 데이터 크기를 계산한 후, 그 대역폭에 맞게 전송하는지 파악한다. 만약 대역폭에 맞지 않다면 그 패킷이 지연되어야 할 지피 수를 계산하여 timestamp 에 반영한다. 이렇게 계산된 지연시간은 큐 지연시간과 함께 해당 패킷의 전송시간을 결정한다. 큐 지연시간은 사용자가 지피값으로 설정할 수 있으며 이 값은 해당 패킷의 timestamp에 더해진다.

사용자는 대역폭과 큐 지연시간 뿐만 아니라 손실률을 설정할 수 있는데, 손실률은 모든 패킷이 전송되기 전 단계인 디바이스 제어 부분에서 계산된다. 디바이스에서는 패킷을 전송할 때마다 손실률 만족 여부를 파악하여 만족하지 않는다면 그 패킷을 버리게 된다.

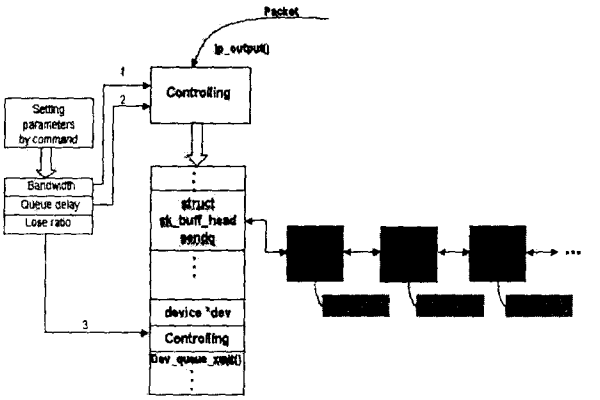

그림 2 NET의 데이터구조

<그림 2>는 NET의 데이터구조를 설명하고 있다. 사용자는 대역폭, 큐 지연시간, 손실률 등을 정의해 줄 수 있고, 패킷은 소켓버퍼로 들어 올 때 대역폭, 큐 지연시간에 의해 제어를 받는다. ip_output() 내에서 대역폭에 의한 패킷의 timestamp 는 다음과 같이 정의된다.

```
sk->timestamp = sk->len / LEN_PER_JIFFIE + jiffies ;
sk->timestamp = sk->timestamp + QUEUE_DELAY ;
```

위에서 생성된 소켓버퍼의 timestamp는 패킷을 송신할때 가장 중요한 비교 요소가 된다. 즉 모든 조건이 만족하더라도 timestamp의 요건이 충족되지 않으면 송신은 유보된다. timestamp가 덧붙여져 큐에서 대기하는 각각의 소켓버퍼는 시스템 클럭이 지나는 동안 체크되어 해당 패킷이 나갈 시간이 되면 현재 유효한 디바이스를 통해 전송을 시도한다.

디바이스에서는 패킷을 전송할 때마다 손실률을 계산한다. 즉, 패킷을 송신할 때마다 설정된 손실률 기준치를 만족할 때만 송신하고, 그렇지 않다면 그 패킷은 송신하지 않는다.

3. 실 험

본 절에서는 실험을 통해 앞에서 설계하고 구현한 NET 가 사용자가 원하는 다양한 형태의 네트워크 토폴로지를 에뮬레이션 할 수 있음을 보여주고 있다.

NET를 시험한 네트워크 토폴로지는 <그림 3> 과 같다. 각 노드들은 6Mbps 링크로 연결되어 있고, 노드 1 이 NET를 적재하고 있다. 본 실험에서는 노드 1 에서 노드 2, 노드 4, 또는 노드 5 로 ftp 전송을 시도하였으며, 파일의 크기는 2Mbytes, 23.4Mbytes, 100Mbytes 그리고 411.49Mbytes의 파일들을 전송하여 시험해 보았고 23.4Mbytes의 파일을 성능 비교 자료로 활용하였다.

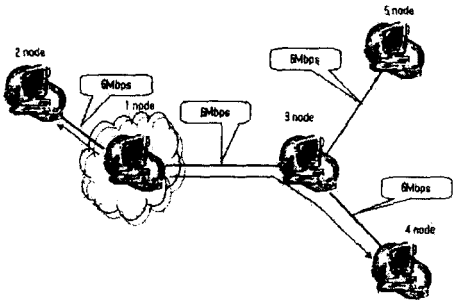

그림 3 네트워크 토폴로지

3.1 사용자의 설정에 관한 실험

사용자가 네트워크 설정을 전혀 하지 않았을 때, 위에서 표현된 네트워크 토폴로지의 평균 대역폭은 약 5.6Mbps 이며 표준편차는 1553.59 Kbps 로 상당히 크다.

<그림 4> 에서는 사용자가 64Kbps 의 대역폭 조건을 주었을 때와 64Kbps 의 대역폭 조건과 함께 5 지피의 큐 지연시간을 함께 주었을 때의 그래프를 보여주고 있다. 사용자가 대역폭의 한계를 주었을 때 약간의 편차가 보이기는 하지만, 사용자가 제한한 한계인 64Kbps 의 조건을 평균 전송속도 60.66Kbps 로 만족시킴을 알 수 있다. 또한 그래프에서 보는 바와 같이 설정된 대역폭에 수렴함을 알 수 있다. 사용자가 설정을 전혀 하지 않은 네트워크와 비교해 볼 때 표준편차가 5.42Kbps 으로서 대역폭의 변화가 줄어들었음을 보인다. 따라서 구현된 NET 는 주어진 대역폭의 조건하에서 안정적으로 동작함을 알 수 있다.

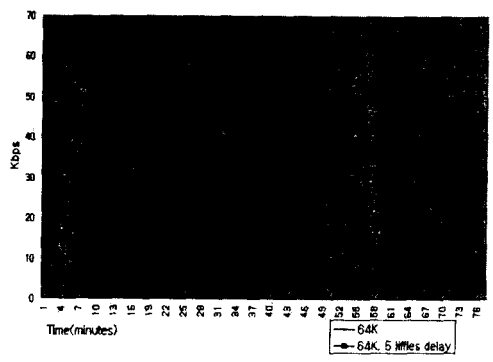

그림 4 대역폭 및 큐 지연시간에 따른 성능

<그림 4> 에서는 대역폭 조건과 함께 큐 지연시간이 주어졌을 때의 성능도 함께 보여주고 있다. 큐 지연시간이 늘어났기 때문에 설정된 64Kbps 의 대역폭보다 줄어든 평균 54.34Kbps 의 대역폭을 보이며 전송이 된다. 큐 지연시간이 함께 주어진 상태의 그래프에서 표준편차는 4.74Kbps 이다. 이것도 사용자가 설정을 전혀 하지 않았을 때의 표준편차보다 안정적으로 동작함을 볼 수 있다.

손실률에 대해서 NET 는 설정된 손실률 조건에 따라 정적으로 동작한다. 이는 평균적으로 보았을 때에는 정확한 평균적 손실률을 갖게 된다. 그러나 실제 네트워크에서는 정적으로 손실이 일어나지는 않는다. 언제 손실이 발생할 것인가는 모르지만 전체 손실률이 설정된 손실률을 만족시켜야 한다. 따라서 패킷 손실은 비 정규적으로 발생하여야 하며 이는 난수 발생 함수와 지수 함수를 구현하여 추후에 추가 시키도록 한다.

4. 요약 및 향후 과제

본 논문에서는 하나의 물리적인 네트워크 토폴로지에서 여러 형태의 토폴로지를 에뮬레이션 할 수 있는 NET를 설계하고 구현하였다. NET 는 커널수준에서 구현되었으므로 응용수준에서 구현된 다른 툴이 가진 문제점들을 갖고있지 않다. 또한 사용자는 NET를 이용하여 대역폭, 큐 지연시간 그리고 손실률을 직접 설정할 수 있고, NET는 사용자의 요구대로 하나의 물리적인 네트워크에서 여러 토폴로지를 에뮬레이션 할 수 있으며 안정적인 동작을 할 수 있음을 보였다.

만약 어떤 네트워크에서 문제점이 발생한다면 이를 해결하기 위해서는 문제 발생시의 네트워크 상태를 정확하게 재연해 줄 수 있는 에뮬레이터가 필요하다. 본 연구자는 지금까지 구현한 NET를 특정 네트워크의 대역폭이나 큐 지연시간, 그리고 손실률을 기록한 Trace파일을 입력 값으로 받아서 해당 시간동안의 그 네트워크 토폴로지 상태를 에뮬레이션 할 수 있도록 발전시킬 것이다. 이는 네트워크 문제발생시 그 해결점을 쉽게 찾고 그 네트워크를 보완시키는데 유용하게 사용되리라 생각한다.

또한 손실률에 있어서도 실제 네트워크에서는 예측 불가능하게 손실이 일어나므로 난수발생 함수와 지수 함수를 구현하여 설정된 손실률을 만족시켜주고 실제 네트워크와 같이 예측 불가능한 손실이 발생될 수 있도록 할 것이다.

5. 참고 문헌

[1]. Nigel Davies, Gordon S. Blair, Keith Cheverst and Adrian Friday, A network emulator to support the development of adaptive applications, April 1995

[2]. David B Ingham, Graham D Parrington, Delayline : A wide-area network emulation tool, 1995.

[3]. Jong Suk Ahn, Peter B. Danzing, Zhen Liu and Limin Yan. Evaluation of TCP Vegas:Emulation and Experiment. SIGCOMM 95 Cambridge, MA USA.

[4]. Jong Suk Ahn. Simulation Performance vs Simulation Timing Granularity. August 1995.

[5]. Alessandro Rubini , " Linux Device Drivers " , O' Reilly, 1998

[6]. M Beck, " Linux Kernel Internals" , Addison Wesley, 1998