리눅스 Bonding 드라이버의 성능 분석

(Performance Analysis of Ethernet Bonding Driver in Linux)

김재열,강동재, 김수영, 차규일

한국전자통신연구원 시스템소프트웨어연구팀

{gauri, djkang, sykim, gicha}@etri.re.kr

요 약

최근의 컴퓨터 시스템은 네트워크를 제외하고는 생각할 수 조차 없을 정도로 네트워크에 대한 의존성이 매우 크다. 저렴한 비용으로 서버시스템에 장착된 다수의 물리적인 이더넷 인터페이스를 하나의 논리적인 이더넷 인터페이스로 만들어 서버시스템의 네트워크 대역폭을 넓히는 방법이 Ethernet Link Aggregation(이하 ELA 라고 칭함)이다. 본 논문은 공개 소프트웨어 중 가장 널리 사용되고 있는 리눅스의 최신 커널 2.6에서 최근의 서버시스템에서 일반적으로 사용하는 기가비트 이더넷 인터페이스 장치를 통해 ELA 기능을 제공하는 리눅스 ethernet bonding 드라이버의 TCP 와 UDP 성능을 측정하여 이를 평가하고 분석하였다.

Keywords : Ethernet Link Aggregation, ethernet bonding, trunking, Linux

1. 서론

최근 멀티미디어 스트리밍 서버에서의 대용량 데이터 전송이나 유비쿼터스 환경의 센서 네트워크에서와 같은 예측할 수 없는 다수 데이터의 전송 요구가 증가함에 따라서 네트워크의 성능(high performance)[1] 및 가용성(high availability)[2]에 관련한 많은 연구들이 진행되어 왔다.

일반적인 2-Way 또는 4-Way 서버처럼 다수의 프로세서를 내장하고 있는 시스템의 경우, 네트워크 처리에 사용할 수 있는 CPU 의 컴퓨팅 자원이 여유 있음에도 불구하고 서버의 물리적 네트워크 인터페이스의 대역폭 제약으로 인하여 네트워크 성능이 제한되는 경우가 발생한다. 이와 같은 경우 서버 시스템의 네트워크 대역폭 확보는 빠른 네트워크 전송이 중요시되는 시스템에서 매우 필수적인 기능이다. 서버의 물리적 네트워크 인터페이스의 대역폭을 증가시키는 가장 쉬운 방법은 더 넓은 대역폭을 지원하는 네트워크 인터페이스 카드와 이를 지원할 수 있는 네트워크 스위치 등의 인프라를 갖추는 것이다. 예를 들면 현재 서버에서 일반적으로 사용하는 기가비트 이더넷을 '10-기가비트' 이더넷 환경으로 전환하는 것이다. 하지만, 이러한 방법은 네트워크 인프라를 바꾸어야 하기 때문에 상당한 비용을 지불해야 한다. ELA는 이러한 추가적인 비용의 지출 없이 현재의 네트워크 인프라를 그대로 이용해 시스템의 네트워크 대역폭을 증가시킬 수 있는 방법으로 매우 유용하다.

다양한 데이터의 서비스를 수행하는 서버들의 경우, 네트워크 디바이스나 기타 회선의 문제로 인하여 발생 가능한 최소한의 네트워크 가용성을 보장하는 것이 요구된다. 일반적으로 ELA 에서는 다수의 물리적인 네트워크 인터페이스 중 일부가 고장이 나거나 네트워크 연결이 끊기는 경우 나머지 네트워크 인터페이스만으로 논리적 네트워크 인터페이스를 구성해 사용할 수 있도록 하는 기능도 제공해 네트워크 가용성을 향상 시킬 수 있다.

ELA 는 여러 개의 물리적 네트워크 인터페이스 들을 하나의 논리적 네트워크 인터페이스로 가상화하는 기술로서 논리적 네트워크 인터페이스에 대한 전송 처리 요구를 하부의 여러 물리적 네트워크 인터페이스들을 통하여 부하분산을 수행 함으로서 네트워크의 대역폭를 증

가시킨다. 따라서, 서버에 대한 클라이언트의 요청에 대한 응답시간을 개선할 수 있으며 네트워크 인터페이스에 대한 대기 시간을 감소시킴으로써 단위 전송의 빠른 처리를 제공한다. 또한, 여러 개의 물리적 네트워크 인터페이스들 중의 일부에 발생하는 고장에 대하여 응용프로그램들에게 투명성을 제공함으로써 네트워크 서비스의 가용성을 향상시킨다.

본 논문에서는 ELA 의 개발 배경과 ELA 가 사용되는 경우의 모델에 대하여 알아보고, 리눅스의 ELA 드라이버인 리눅스 bonding 드라이버[3]에 대하여 살펴본다. 또한 TCP 및 UDP 를 사용하는 응용프로그램 환경에서 벤치마크 실험을 통하여 리눅스 bonding 드라이버의 성능 평가 및 분석을 수행하고 이에 대한 결론과 향후 과제를 도출한다.

2. Ethernet Link Aggregation

본 장에서는 먼저 ELA 의 개발 배경과 ELA 가 사용되는 환경에 대해서 간략히 기술하고, 리눅스의 bonding 드라이버에 대한 구조 및 지원 모드에 대하여 설명한다.

2.1 Ethernet Link Aggregation 의 배경

최초의 ELA 기능은 trunking 이라는 이름으로 각 네트워크 장비업체들이 독자적으로 가지는 특화된 기술로서, 10/100Mbps 이더넷과 FDDI 환경에 적용되어왔다. 예를 들면 네트워크 장비 업체중에는 CISCO 의 ISL(Inter-Switch Link trunking)[4]과 Adaptec 의 Duralink port aggregation[5]이 대표적이며 서버업체로는 Sun 의 Sun Trunking[6]기술이 있다. 그러나 각 업체들마다 서로 다른 프로토콜들은 타 제조사들의 제품에 호환성을 제공하지 못하는 이유로 많은 네트워크 장비 관리자들이 장비 통합에 어려움을 겪게 되었다. 이런 trunking 기술의 타 제조사 장비간 비호환성을 개선하기 위해 나온 것이 IEEE802.3ad[7,8] 표준으로 1999 년에 제정되었다.

ELA 기술은 초기에 스위치나 라우터등의 네트워크 장비들간의 기간 망 연결에 주로 사용되었으나 수 년전부터 서버 시스템의 네트워크 대역폭의 중요성이 커지면서 현재는 서버와의 연결 지원도 매우 중요하다.

ELA 를 적용함에 있어서는 크게 아래와 같은 3 가지 모델로 정리가 된다. [그림 1]의 첫 번째 모델은 ELA 가 처음으로 사용될 때의 일반적인 모델로 스위치와 스위치간을 연결한다.

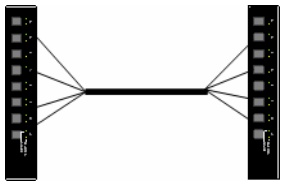

그림 1. 스위치와 스위치간 ELA 연결

[그림 1]에서와 같이 초창기에는 스위치와 같은 통신장비들 사이의 연결에만 ELA 기능이 적용되었다. 하지만, 서버의 네트워크 기능이 점차 증가함에 따라 [그림 2]에서와 같이 서버와 스위치간에도 EAL 가 사용되었다. 물론 이때 사용되는 스위치는 ELA 기능을 지원하는 스위치여야 한다.

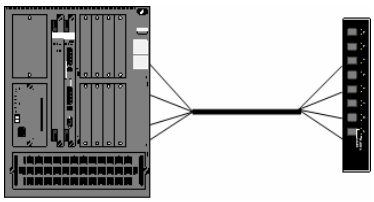

그림 2. 서버와 스위치간 ELA 연결

이러한 구성과 더불어 최근에는 서버와 서버의 네트워크 연결에도 ELA 를 적용할 수 있다. [그림 3]은 이러한 연결을 보여준다. [그림 3]과 같은 경우는 여러 대의 서버가 동일한 서비스를 하는 경우와 같이 근접한 거리에 서버들이 모여있을 때 적용할 수 있다. 이 기종으로 이루어진 Beowulf cluster 에도 이러한 ELA 를 적용한 사례[9]가 있다.

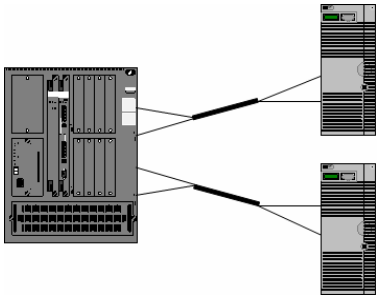

그림 3. 서버와 서버간 ELA 연결

2.2 리눅스 bonding 드라이버의 구조

ELA 는 물리적인 다수의 네트워크 인터페이스들을 하나의 논리적인 네트워크 인터페이스로 가상화하고, 논리적인 네트워크 인터페이스에 대한 데이터 전송 및 수신 요청을 bonding 에 참여하는 실제 네트워크 인터페이스(이하 “slave”로 기재)들에게 다양한 로드 분산 알고리즘에 의하여 처리를 분산한다.
이러한 다양한 알고리즘에 따른 분류를 모드라고 하며, 각 모드에 관한 설명은 다음절에서 기술하도록 하며 본 절에서는 ELA 를 위한 리눅스 bonding 모듈의 구조에 대하여 설명한다.
[그림 4]는 ethernet bonding 모듈의 커널내에서의 위치 및 간략한 구조를 보여주는 그림이다. bonding 모듈은 실제 물리적인 디바이스를 운용하기 위한 드라이버의 가상화를 위한 부분이므로 시스템에서 사용하는 네트워크 디바이스 드라이버의 상위에 존재한다.

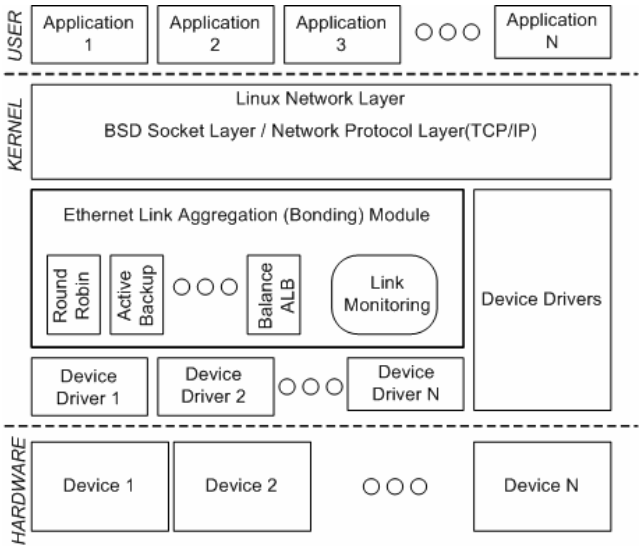

그림 4. Bonding 드라이버의 구조와 커널의 관계

[그림 4]에서의 link monitoring 부분은 bonding 에 참여하는 각 slave 의 가용성을 주기적으로 점검하는 부분으로써 일부 slave 에 대한 고장이 인식되면 현재의 bonding 설정을 변경한다. 본 기능은 디바이스 드라이버의 인터페이스에서 MII(Media Independent Interface)나 환경설정 도구인 ethtool[10]이 제공하는 경우에 사용 가능하다. 예를 들면, 현재 사용중인 slave 가 고장으로 인식되면 bonding 에 참여하는 가용한 다른 slave 를 현재 사용을 위한 slave 로 재지정하는 동작을 수행하게 된다. [그림 4]에서 round robin 과 같은 부분들은 bonding 모듈을 통한 전송 요청 시에 데이터를 어떠한 기준으로 하부 slave 들에게 분배할 것인지를 결정하기 위한 정책들을 구현한 부분이며 자세한 내용은 다음 절에서 설명하도록 한다.

bonding 모듈이 커널에서 사용되면, bonding 에 참여하는 모든 slave 들과 논리적인 bonding 네트워크 인터페이스는 외부에 공개되는 동일한 IP 와 MAC 주소 및 설정을 공유하게 되어서 통신상대가 되는 원격의 노드와 응용프로그램에게 bonding 에 대한 투명성을 제공한다.

2.3 리눅스 bonding 드라이버의 지원 모드

본 절에서는 리눅스의 bonding 드라이버에서 사용 가능한 모드에 대하여 살펴본다. 지원하는 모드들의 목적은 고성능을 위한 load-balancing 부분과 고가용성의 보장을 위한 fault tolerance 로 구분할 수 있으며 모드에 따라서 일부 또는 동시 지원을 하고 있다.

Balance-Round Robin Mode.

현재 bonding 을 위하여 slave 로 등록된 NIC(Network Interface Card)의 처음부터 마지막까지 순차적으로 데이터를 전송하는 방식으로 본 모드에서는 네트워크에 대한 전송 부하를 여러 개의 slave 로 분할하여 전송함으로써 load balancing 을 통한 네트워크 대역폭 증가 효과를 얻을 수 있다. 또한 slave 로 등록된 일부 NIC 의 고장 시, 나머지 NIC 으로 전송을 수행함으로써 네트워크 가용성 또한 보장한다.

Active-Backup Mode.

Bonding 을 위하여 등록된 slave 중 오직 하나의 NIC 만이 active 상태이며 나머지는 backup 을 위하여 inactive 상태로 존재한다. 현재 active 상태인 NIC 에 고장이 발생하는 경우, backup 을 위하여 등록된 나머지 NIC 중에서 하나가 active 로 상태 전이를 하게 됨으로써 끊임없는 데이터 전송 처리를 수행할 수 있도록 하는 모드이다. 고장이 발생한 NIC 에 대한 backup 시에는 동일한 서비스 port 에 대하여 동일한 MAC 어드레스가 외부에 공개됨으로써 어플리케이션들에 대한 고장 상태의 투명성을 제공한다. 이 모드에서는 대역폭의 증가 효과는 없으며 네트워크의 고가용성만을 보장한다.

Balance-XOR Mode.

데이터 전송 시에 [destination MAC address ⓧ source MAC address **%** slave count]의 연산에 근거하여 사용할 slave 를 선택하는 모드이다. 따라서, 동일한 목적지에 대한 데이터 전송인 경우, 동일한 slave 를 통하여 수행된다. 본 모드에서도, Balance-Round Robin 모드와 동일하게 load balancing 과 fault tolerance 기능을 동시에 지원한다.

Broadcast Mode.

slave 로 등록된 모든 NIC 에 대하여 데이터 전송을 중복 처리하는 모드이다. 동일한 데이터에 대하여 slave 로 등록된 NIC 에 대하여 중복 전송을 수행하므로 처리 부하가 크다. 따라서, 단일 NIC 의 사용시 보다 성능은 감소하지만 slave 로 등록된 모든 NIC 에 고장이 발생하지 않는한 네트워크 전송은 수행되어 우수한 fault tolerance 를 보장한다.

802.3ad Mode.

본 모드에서는 slave 들을 동일한 속도와 양방향 통신의 설정을 공유하는 aggregation group 으로 통합하며 active aggregator 에 소속된 모든 slave 를 통하여 송수신을 수행하는 모드이다.

Balance-TLB(Adaptive Transmit Load Balancing).

데이터의 송신은 현재 slave 로 등록된 각 NIC 에 대한 load 상태의 계산에 의하여 분산되며 데이터의 수신은 현재 사용중인 slave 에 의하여 처리되므로 수신에 대한 load balancing 은 지원하지 않는다. 수신을 위한 slave 에 고장이 발생하는 경우, 다른 slave 가 고장이 발생한 slave 의 MAC 주소를 인계 받아서 처리를 수행한다.

Balance-ALB(Adaptive Load Balancing).

상기 Balance-TLB 모드의 수행 방식에 수신을 위한 load balancing 기능이 추가된 모드이며 수신을 위한 load balancing 은 ARP 협상에 의하여 수행된다. 서버에 의하여 발생된 ARP 응답 메시지를 bonding 모듈이 hooking 하여 ARP 응답메시지에 존재하는 발신 측의 하드웨어 주소를 bonding 에 참여하는 임의 slave 의 유일한 하드웨어 주소로 덮어쓰기 연산을 수행함으로써 bonding 에 참여하는 모든 slave 는 각자 다른 하드웨어 주소를 사용하게 된다. 따라서, 수신 시에 각자 해당하는 데이터 패킷을 수신함으로써 수신을 위한 load balancing 기능을 제공한다.

시스템 관리자는 시스템의 환경에 따라서 상기 모드들의 ELA 중 적절한 것을 선택해야 한다. 일반적으로 간단한 환경에서는 처리 방식이 단순한 balance round-robin 의 경우가 좋은 성능을 나타낼 수 있다.

3. 성능 분석 및 평가

3.1 실험 환경

리눅스 커널 2.6 에 포함되어 있는 ELA 의 성능을 테스트하기 위해 아래와 같은 환경에서 실험을 하였다. 실험에는 리눅스 bonding 드라이버의 여러 모드중 일반적으로 가장 성능이 좋은 balance round-robin 모드를 사용하였다.

실험에 사용된 장비는 서버와 스위치로 이루어 지며, 서버는 모두 동일한 사양을 가지고 있다.

서버에는 Intel Xeon 3.0GHz 의 CPU 가 2 개씩 장착되어 있으며, 메모리는 각각 1GB 가 장착되어 있다. 실험에 사용된 네트워크 카드는 서버 보드에 내장되어 있는 인텔의 e1000 기가비트 이더넷 포트 2 개를 사용하였다. 서버에 사용된 커널은 리눅스 커널 2.6.10 이다.

스위치는 3com 의 3c17700 기가비트 스위치를 사용하였으며 네트워크 성능을 측정하는 benchmark tool 로는 널리 알려진 SGI 의 netperf 2.1 을 사용하였다. netperf 는 테스트 시간을 지정할 수 있으며, 이를 30 초로 지정해 모든 실험을 진행하였다.

TCP 실험에서는 대부분 netperf 의 default 설정을 이용하여 실험하였으며 몇몇 옵션을 지정해 실험한 경우도 있었으나 결과값이 크게 다른 경우가 없어 본 논문에서 따로 결과를 보여주지는 않는다.

아래는 본 실험에 사용된 netperf 의 tcp 테스트 옵션이다.

- netperf –l 30 –H 192.168.1.1

UDP 실험에서는 소켓버퍼의 크기와 message 의 크기를 16060 byte 로 지정하여 테스트하였다.

아래는 본 실험에 사용된 netperf 의 udp 테스트 옵션이다.

- netperf –l 30 –t UDP_STREAM –H 192.168.1.1 -- -s 16060 –m 16060

3.2 실험 A (2G bonding - Switch – 2G bonding)

실험 A 는 [그림 5]와 같은 환경에서 수행되었다. 이 실험의 목적은 ELA 의 기능과 성능을 측정하는 것을 목적으로 수행되었다.

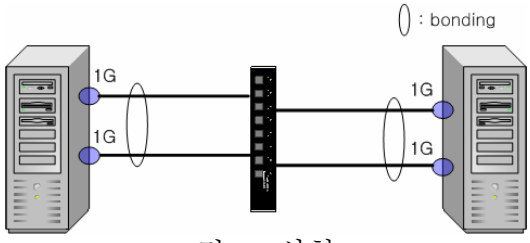

그림 5. 실험 A

[그림 5]에서 보듯이 첫 번째 실험은 두 서버의 양단을 mode-0(round-robin)으로 bonding 시킨 상태에서 스위치에 연결하여 측정 하였다. 리눅스의 bonding 이 제공하는 mode-0 에서 mode-6 까지의 7 개의 mode 중 mode-0(round-robin)를 선택하여 실험한 이유는 [그림 5]와 같은 가장 기본적인 네트워크 환경에서 link aggregation 기능을 제공해 최대의 성능을 낼 수 있는 mode 가 round-robin 이기 때문이다.

실험은 TCP 와 UDP 모두에 대하여 수행하였으며 프로세스의 개수에 따른 성능상의 차이점을 알아보기 위해 테스트 프로세스(connection) 개수를 1, 5, 10, 20 로 바꾸어 가면서 테스트하였으며 bonding 을 하지 않았을 때의 성능을 비교해 보기 위해 단일 이더넷 카드(1-NIC)의 경우에도 실험을 수행하였다.

[표 1] 실험 A 의 결과 (단위 : Mbps)

	TCP		UDP	
Stream	1-nic	Bond(r-r)	1- nic	Bond (r-r)
1	941.1	941.04	960	1918.59
5	942.22	970.45	-	-
10	939.65	941.99	-	-
20	935.17	982.12	-	-

[표 1]의 결과를 살펴보면 먼저, 1-NIC 의 경우에는 기가 비트 카드의 최대성능인 1Gbps 에 근접하게 나왔으며 TCP 나 UDP 가 큰 차이가 없음을 알 수 있다. 약간의 성능상의 차이점은 TCP 가 UDP 보다 connection based 방식으로 운용되기 때문에 생기는 부하 때문이라고 추정된다.

Bonding 을 한 서버간의 테스트 결과를 살펴보면 TCP 의 경우에는 2 개의 link 를 aggregation 했음에도 불구하고 1Gbps 를 넘지 못한 결과를 보여주고 있으며, UDP 의 경우에는 2Gbps 에 가까운 좋은 성능을 보여주었다. UDP 의 경우에는 1-NIC 경우의 결과를 표시하고 다수 프로세스의 결과는 기록하지 않았다.

netperf 는 자체적으로 다수의 process 를 수행해 네트워크 bandwidth 를 측정하는 기능은 제공해 주지 않기 때문에 본 실험에서 다수의 프로세스가 네트워크를 이용한 결과를 측정할 때는 복수의 netperf 프로그램을 수행시키는 스크립트를 작성하여 사용하였다. 그러나 이런 방식을 사용함으로써 생기는 문제는 process 를 의도대로 항상 동시에 수행시키지 못할 수 있다는 것이다. TCP 의 경우에는 대부분 다수의 프로세스가 동시에 수행되었으나 UDP 의 경우에는 각 프로세스 사이의 간격이 TCP 에 비해 매우 커서 결과값을 신뢰할 수 없었다. 또한 하나 이상의 UDP 대역폭 실험이 큰 의미를 갖지 않는 이유는 결과에서 볼 수 있듯이 단일 프로세스 실험에서도 1900Mbps 이상의 성능을 보여주어 물리적인 한계인 2Gbps 에 가깝게 사용하고 있음을 알 수 있기 때문이기도 하다. 결과적으로 UDP 의 경우는 프로토콜의 특성상 단일 연결에서도 최대의 네트워크 성능을 측정하기에 충분함을 알 수 있다.

위 실험 A 에서 TCP 의 경우 성능이 기대했던 것과는 달리 1Gbps 를 넘지 않았다. 이런 성능상의 문제가 리눅스의 bonding 드라이버 자체의 문제인지, 주위의 네트워크 환경이나 bonding 을 수행하는 서버의 처리 능력이 부족한 것인지를 확인하기 위해 실험 B 를 수행하였다.

3.3 실험 B (200M bonding – Switch – 1G)

실험 A 에서 TCP 성능에 bonding 효과가 전혀 나타나지 않은 이유를 찾기 위하여 실험 B 에서는 send 를 하는 서버의 주위 네트워크 환경이 충분히 넓은 대역폭을 제공해 줄 수 있도록 송신 서버에 기가 비트 카드가 아닌 100Mbps 이더넷 카드 2 장을 설치해 이를 bonding 으로 묶어 테스트를 진행하였다. [그림 6]은 실험 B 의 실험 환경을 보여 준다.

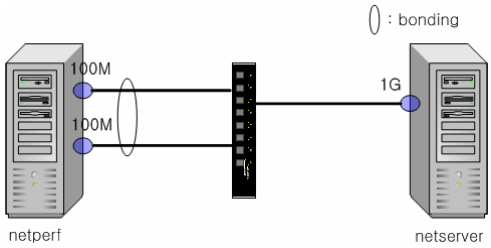

그림 6. 실험 B

[표 2]는 실험 B 의 실험 결과를 보여준다. 실험 B 의 결과를 살펴보면 UDP 의 경우에는 192Mbps 로 실험 A 의 결과처럼 bonding 된 네트워크 대역폭을 충분히 활용하고 있으며 TCP 의 경우에도 188Mbps 라는 좋은 대역폭을 보여주었다.

[표 2] 실험 B 결과 (단위 : Mbps)

	TCP		UDP	
Stream	1 NIC	Bond (r-r)	1 NIC	Bond (r-r)
1	94.13	145.12	96.13	192.12
5	94.22	188.34	-	-
10	94.35	188.91	-	-
20	94.27	188.45	-	-

이러한 결과에서 추정해 볼 수 있는 것은 bonding 드라이버가 하위의 이더넷 인터페이스 카드의 scalability 를 제대로 지원해주지 못하는 것은 아니라고 추정할 수 있으며 따라서 실험 A 에서 TCP 의 성능이 잘 나오지 않은 이유는 송신 호스트의 processing 능력이 2Gbps 의 네트워크 부하를 제대로 처리하지 못하거나 원격 호스트의 수신 처리 능력이 떨어진다고 볼 수 있다. 일반적으로 TCP/IP 네트워크 처리에 드는 CPU 부하는 1bps 처리에 1hz 정도가 소요된다고 알려져 있으며 실험에 사용한 서버의 CPU 는 Intel Xeon 3.0Ghz 두 개를 사용한 SMP 시스템으로 2Ghz 의 네트워크 부하를 처리하는 데는 무리가 없을 것으로 판단되므로 실험 A 에서 TCP 의 성능이 제대로 나오지 않은 것은 수신 측의 bandwidth 가 충분하지 않기 때문일 것으로 추정할 수 있다.

이와 같은 추정을 확인하기 위해 수신 측의 bandwidth 를 충분히 확보한 실험 C 를 수행하였다.

3.4 실험 C (2G bonding – Switch – 1G x 2)

실험 C 는 수신 측의 대역폭을 충분히 확보하기 위해 netperf test 의 수신 측 서버를 한대의 호스트가 아닌 두 대의 호스트에서 처리하도록 [그림 7]와 같은 환경하에서 수행하였다.

실험 C 시험환경의 특징은 netperf client 의 데이터를 받는 netserver 를 두대의 호스트에서 수행함으로써 netserver 의 수신 대역폭을 보장할 수 있는 것이다. netperf client 는 서로 다른 두 개의 netserver 로 각각 데이터를 보내 client 의 최대 대역폭을 확인할 수 있다. 이전의 실험에서 TCP 의 최대 대역폭을 측정하는데 10 개 이상의 process 이면 충분하다는 것을 확인할 수 있었으므로 실험 C 에서는 TCP 와 UDP 모두 각 netserver 로 5 개의 요청을 보내, 총 10 개의 process 를 수행하는 것으로 실험을 하였다.

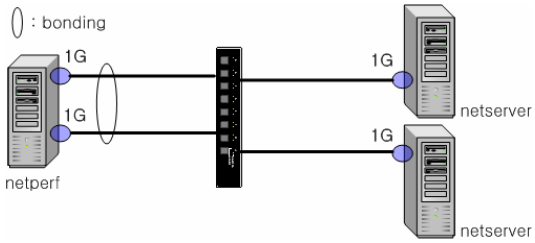

그림 7. 실험C

[표 3]은 실험 C 의 결과를 보여준다. 실험 C 의 결과를 보면 실험 A 와는 달리 TCP 의 경우에도 1800Mbps 에 가까운 만족할 만한 성능을 보이고 있다. 이는 실험 A 에서의 TCP 테스트의 성능이 제대로 나오지 않은 이유가 수신 측의 bandwidth 가 충분하지 않았기 때문이라고 할 수 있다. UDP 테스트 결과는 실험 A 에서와 마찬가지로 충분한 성능을 보이고 있다.

이의 실험 결과를 바탕으로 실험에 사용된 리눅스 커널의 bonding 드라이버는 send 시에 충분한 bandwidth 를 제공하고 있다고 판단할 수 있다.

[표 3] 실험 C 결과 (단위 : Mbps)

	TCP(5+5)	UDP(5+5)
#	Bond(r-r)	Bond(r-r)
1	1789.17	1924.77
2	1778.56	1935.21
3	1790.16	1922.15
4	1785.35	1900.37

그렇다면 문제가 되는 것은 실험 A 의 bonding 시스템에서 receive 대역폭이 확보되지 않았다는 점이며 이를 테스트 하기 위하여 실험 D 를 수행하였다.

3.5 실험 D (2G bonding – 2G bonding)

리눅스 bonding 드라이버의 send 시의 성능은 실험 C 에서 검증되었으므로 이제 bonding 드라이버가 receive 시에 얼마만큼의 대역폭을 처리할 수 있는지를 테스트하기 위해 [그림 8]과 같이 각각 bonding 한 호스트를 크로스 케이블을 사용해 스위치를 거치지 않고 직접 연결하였다. 이렇게 하면 실험 C 에서 검증된 bonding 드라이버의 TCP send 성능이 1.8Gbps 의 속도를 낼 수 있을 것이므로 이 실험의 결과치가 1.8Gbps 이하로 나온다면 이를 bonding 드라이버의 receive 성능이라고 볼 수 있을 것이다.

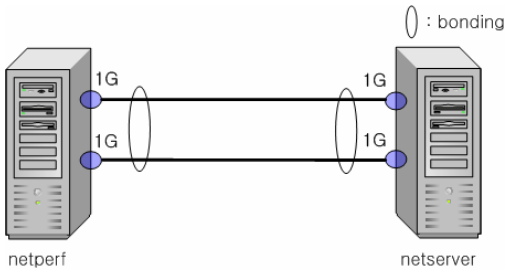

그림 8. 실험 D

이러한 구성이 가능한 이유는 mode-0, round-robin 으로 bonding 을 구성한 경우에 send 측에서는 각 패킷마다 round-robin 방식으로 하위의 다수의 slave 로 번갈아 가면서 보내기 때문에 두 대의 호스트를 직접 연결한 경우에는 스위치의 도움을 받지 않고 receive 측에서는 차례대로 packet 을 받을 수 있기 때문이다.

TCP 의 경우 각 connection 을 갖는 process 를 5 개, 10 개씩을 사용하여 실험한 결과를 보여준다. 10 개 이상이 되었을 때는 프로세스 개수가 최대 bandwidth 측정에 영향을 주지 않아 10 개 까지만 테스트하였으며, UDP 의 경우에는 실험 1 에서 언급한 것과 같이 하나의 process 에서 최대 bandwidth 를 낼 수 있으므로 process 개수는 1 개일 경우만 실험하였다.

앞의 실험에서 언급하지 않은 netperf 의 결과 중 하나는 UDP 성능 측정의 경우 netperf 는 send 시의 성능과 receive 의 성능을 함께 측정해 준다. 예를 들면 [그림 9]은 netperf UDP 테스트의 출력 결과의 한 예를 보여준다.

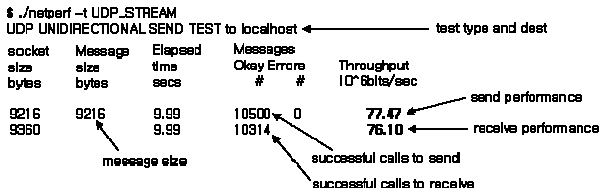

그림 9. netperf 의 UDP 시험 결과 예

[그림 9]에서 테스트의 결과는 send 시와 receive 시의 두 개가 출력된다. UDP 는 TCP 와는 달리 수신 측에서 송신한 데이터를 제대로 받았는지, 그렇지 않은지를 알 수도 없으며 상관하지 않는다. 그래서 UDP 테스트의 경우에는 send 측에서 보낸 UDP 데이터를 receive 측에서 에러 없이 잘 받았는지를 나타내는 receive 성능을 함께 보여준다. 실험 D 는 TCP 와 UDP 의 수신 성능을 측정하기 위한 실험이므로 netperf 의 UDP 테스트 수신 성능 또한 유효한 의미를 가지고 있다.

[표 4] 실험 D 의 결과(단위 : Mbps)

Stream	TCP	#	Type	UDP
5	1550.6	1	Send	1921.42
5	1518.07		Receive	1914.53
10	1449.45	2	Send	1920.94
10	1457.71		Receive	1920.87

[표 4]는 실험 D 의 결과를 보여준다. 먼저 TCP 성능 결과를 살펴보면 1450 Mbps 에서 1550 Mbps 정도의 성능을 보여주고 있다. 이는 실험 C 에서 얻을 수 있었던 1.8 Gbps 의 대역폭에 비해 떨어지는 것이므로 이 값을 bonding 드라이버의 TCP receive 성능이라고 볼 수 있을 것이다. 따라서 실험의 결과에 의하면 리눅스의 bonding 드라이버는 TCP 의 경우는 send 시가 receive 보다 더 좋은 성능을 보여준다는 것을 알 수 있다.

UDP 의 경우에는 두 번의 실험을 하였으며 그 결과 send 와 receive 모두 1920 Mbps 의 성능을 보여주고 있어 대부분이 에러 없이 send 와 receive 가 되었으므로 bonding 드라이버의 UDP receive 성능은 send 시의 성능과 동일한 1920 Mbps 정도라고 볼 수 있다. 즉 리눅스의 bonding 드라이버는 UDP 의 경우 send 와 receive 시 둘 다 1.9Gbps 이상의 좋은 성능을 보이고 있음을 확인할 수 있다.

실험 A 에서 TCP 의 성능이 실험 D 에서의

성능에 비해 떨어지는 것은 스위치의 trunking 동작이 정상적으로 동작하지 않았기 때문이라고 추정할 수 있다. 실험 C 와 실험 D 는 스위치를 배제한 호스트간의 송수신 능력을 측정한 결과로 스위치의 성능여부와 상관없는 bonding 드라이버 자체의 성능을 보여준다.

4. 결론 및 향후 계획

본 실험은 리눅스 커널 2.6 에 포함되어 ELA 를 지원하는 리눅스 bonding 드라이버의 기능과 성능을 테스트해 보고 문제가 있다면 문제의 원인을 밝히는 것이 목적이다. 실험에서는 최대한 스위치의 기능과 성능에 따른 영향을 최소화 하도록 실험을 진행하였다.

실험의 결과를 살펴보면 Ethernet link aggregation 기능을 제공하는 mode-0 (round-robin)에서 TCP 와 UDP 모두 충분한 성능을 발휘할 수 있다는 것을 확인할 수 있었다. 2 개의 gigabit port 를 bonding 했을 경우 TCP 는 send 의 경우 1.8Gbps, receive 의 경우 1.5Gbps 의 대역폭을 보였으며 UDP 의 경우 send 와 receive 모두 1.9Gbps 초반의 대역폭을 제공하였다.

이러한 테스트 결과를 바탕으로 리눅스 커널 2.6 에서 제공하는 bonding 드라이버의 link aggregation 은 UDP 의 경우에는 매우 좋은 선택일 수 있으며, TCP 의 경우에도 send 를 주로 담당하는 서버에서 사용한다면 효과적일 수 있을 것이라는 결론을 내릴 수 있다. link aggregation 을 사용할 경우에는 위의 실험결과를 참고하여 환경을 설정할 때 최상의 효과를 얻을 수 있을 것이다.

향후 과제로는 본 논문의 실험에서 다루지 못한 리눅스 bonding 드라이버의 기타 다른 모드의 성능측정이 필요하며, 또한 다수의 네트워크 인터페이스 카드를 사용함에 따른 성능의 변화를 측정하는 scalability 실험도 필요하다. 정확한 리눅스 bonding 드라이버의 scalability 를 안다면 실제로 서버에 이를 적용하는 경우에 매우 유용할 것으로 생각된다.

참고 문헌

[1] Phillip Dykstra, “High Performance Networking”, on SuperComputinf 2002 in 18 November 2002

[2] Deepak Kakadia, Sam Halabi and Bill Cormier, “Enterprise Network Design Patterns: High Availability”, www.sun.com/blueprints/1203/817-4683.pdf

[3] Thomas Davis, Willy Tarreau, Constantine Garvrilov, Chad N. Tindel, Janice Girouard, Jay Vosburgh, “Linux Ethernet Bonding Driver mini-howto”, 2000.

[4] “ISL and 802.1Q Trunking Between Catalyst Layer 2 Fixed Configuration Switches and CatOS Switches Configuration Example”, www.cisco.com/public/473/43.html

[5] “Duralink Port Aggregation Software”, www.adaptec.com/pdfs/portaggregation.pdf

[6] “SUN TRUNKIN 1.2”, White papers in Sun Microsystems, 1999.

[7] “Link Aggregation according to IEEE Standard 802.3ad”, White papers in SysKonnect GmbH, 2002.

[8] “Amendment to carrier sense multiple access with collision detection(CSMA/CD)access method and physical layer specification aggregation of multiple link segments”, IEEE Std 802.3ad-2000

[9] Daniel Andersen and Zhao Baosong, “Heterogeneous Channel Bonding on a Beowulf Cluster”, In Proceeding of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, pp 2479-2484, June 2000.

[10] ethtool, project site, “http://sourceforge.net/projects/gkernel/”

\

김 재 열

1999 경북대학교 전자공학과 학사

2001 경북대학교 전자공학과 석사

2001 ~ 현재 한국전자통신연구원, 선임연구원

<관심 분야> 리눅스 운영체제, 파일 시스템, 시스템 소프트웨어

강 동 재

1999 인하대학교 전자계산학과 학사

2001 인하대학교 전자계산공학 석사

2004 ~ 현재 인하대학교 컴퓨터 정보공학 박사과정

2001 ~ 현재 한국전자통신연구원, 연구원

<관심 분야> 데이터베이스, 네트워크 연결형 자료저장시스템, 원격시스템관리, 리눅스 기반 기술

김 수 영

2003 중앙대학교 컴퓨터공학과 학사

2005 한국과학기술원 전산학과 석사

2005 ~ 현재 한국전자통신연구원, 연구원

<관심분야> 리눅스 커널, 임베디드 시스템 소프트웨어

차 규 일

1998 고려대학교 컴퓨터학과 학사

2000 고려대학교 컴퓨터학과 석사

2000 ~ 현재 한국전자통신연구원, 선임연구원

<관심분야> 운영체제 성능 개선, 시스템 소프트웨어 설계, 분산 컴퓨팅