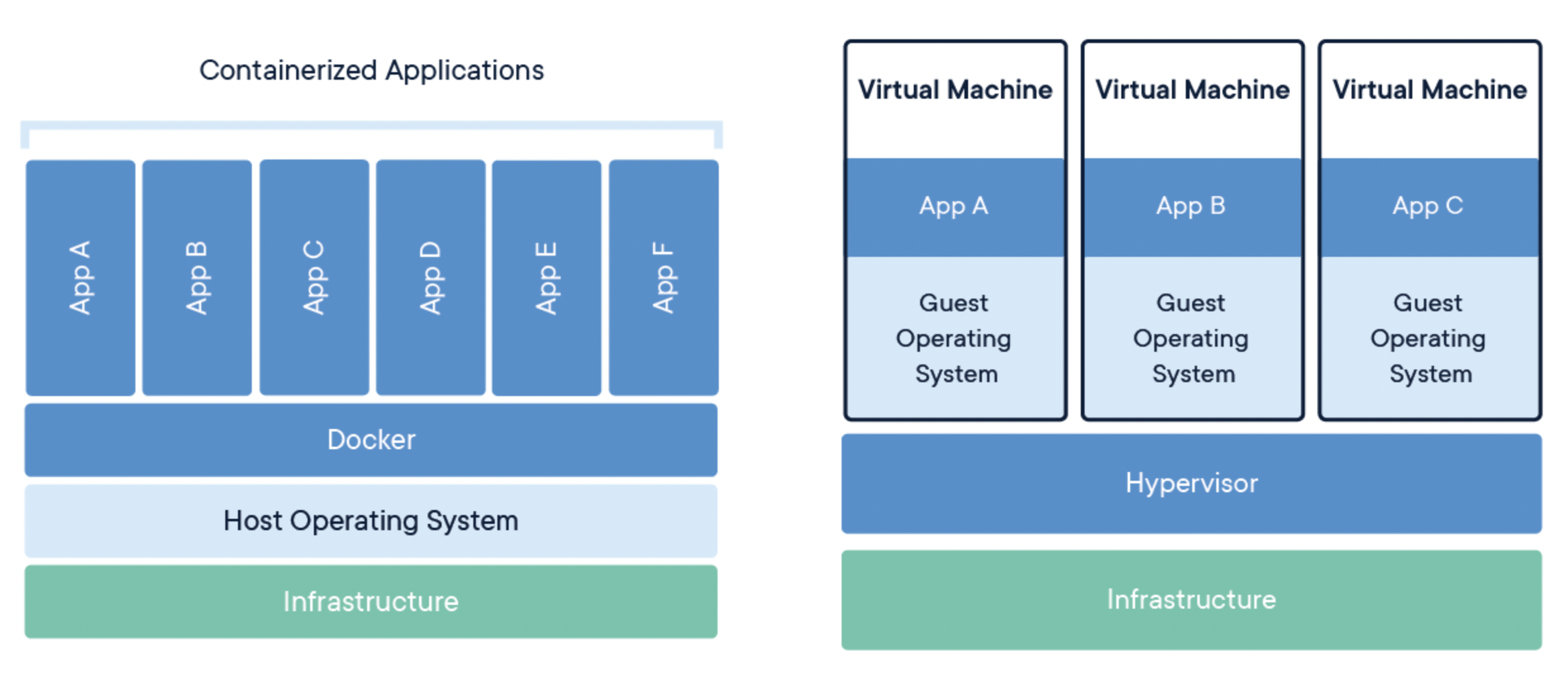
**1. 도커(Docker)란 무엇인가?**

도커는 컨테이너 기반의 오픈소스 가상화 플랫폼이다.

**2. 컨테이너(Container)란 무엇인가?**

도커를 이해하기 위해선 컨테이너에 대한 이해가 선행되어야 한다. 컨테이너는 어떠한 컴퓨팅 환경에서도 독립적으로 실행될 수 있도록 애플리케이션 코드와 연관된 라이브러리 및 종속 항목들을 함께 패키징한 소프트웨어 실행 유닛이다. 이를 위해 컨테이너는 운영체제 수준의 가상화를 활용하여 프로세스를 격리하고 해당 프로세스가 액세스할 수 있는 CPU, 메모리 및 디스크 할당량을 제어한다. 여기서 운영체제 수준의 가상화(operating-system-level virtualization)란 운영 체제의  커널이 하나의 사용자 공간 인스턴스가 아닌, 여러 개의 격리된 사용자 공간 인스턴스를 갖출 수 있도록 하는 것이다.

**3. 도커(Docker)와 가상머신(Virtual Machine) 비교**



(출처: https://www.docker.com/resources/what-container)

도커와 컨테이너 모두 가상화를 통해 어플리케이션을 격리한다는 공통점이 있다. 그러나 도커는 앞서 언급했듯 운영체제 수준에서 가상화를 진행하는 반면, 가상 머신(VM)은 하드웨어를 가상화로 하나의 서버를 다수의 서버로 분할한다. 하이퍼바이저(Hypervisor)는 하나의 머신에서 여러 개의 가상 머신이 실행될 수 있도록 동작한다. 그리고 각각의 가상 머신은 게스트 운영체제를 필요로 한다.

이외의 핵심 비교 사항들을 항목별로 정리하여 아래 표로 나타내었다.

|  |  |  |
| --- | --- | --- |
|  | 도커 | VM |
| 게스트 OS 설치 유무 | 각각의 컨테이너들은 호스트 OS를 공유하기 때문에 별도의 게스트 OS를 필요로 하지 않는다. | VM마다 게스트 OS를 필요로 한다. |
| 격리를 위한 가상화 수준 | 운영체제 | 하드웨어 |
| 생성 소요 시간(컨테이너 또는 가상머신) | 초(second) 단위 | 분(minute) 단위 |
| 부팅 소요 시간 | OS 설치가 불필요하므로 사용 시작까지의 소요 시간이 매우 짧다. | 초기 구축 시 네트워크 OS 설치 등의 작업을 요구하기 때문에 시간이 많이 소요된다. |
| 네트워크 | 호스트 측의 Docker 전용 NIC와 통신 | VM에 임의의 개수의 vNIC를 붙여 통신 |

**3. 네트워크 스위치(Network Switch)란**

네트워크 스위치는 OSI 모델의 데이터 링크 계층(계층 2)에서 작동한다. 물리적 포트에 연결된 기기에서 전송된 패킷을 받아 다시 내보내는데 단, 패킷이 도달해야 하는 기기로 이어지는 포트를 통해서만 보낸다. 현재는 대부분의 스위치가 이더넷을 사용한다.

**4. 네트워크 스위치의 작동 원리**

기기가 스위치에 연결되면 스위치는 해당 이더넷 케이블에 연결된 기기의 NIC에 내장된 MAC 주소를 확인한다. 스위치는 이 MAC 주소를 사용해 패킷이 발송된 기기와 수신된 패킷을 전달할 위치를 식별한다. 따라서 MAC 주소는 기기에 동적으로 할당되고 변경이 가능한 네트워크 계층(계층 3) IP 주소와 달리 물리적 기기를 식별하는 데 쓰인다.  
  
한 기기가 다른 기기로 패킷을 보내면 패킷이 스위치로 들어오고 스위치는 패킷의 헤더를 읽어 어떻게 처리할지 결정한다. 목적지 주소와 대조해 목적지 기기로 이어지는 적절한 포트를 통해 패킷을 내보낸다.  
  
이 과정에서 스위치와 스위치에 연결된 기기에서 동시에 전송되고 수신되는 네트워크 트래픽 간의 충돌 가능성이 있으므로, 대부분의 스위치는 기기를 오가는 패킷이 스위치 연결의 전체 대역폭에 액세스할 수 있도록 하는 전이중(full-duplex) 기능을 제공한다. 대부분의 스위치가 계층 2에서 작동하지만(L2 스위치) 일부 스위치는 계층 3에서도 작동할 수 있다(L3 스위치).

**5. 스위치와 허브 비교**

허브 역시 리소스 공유를 목적으로 여러 기기를 하나로 연결할 수 있는데, 허브에 연결된 기기 집단을 LAN 세그먼트라고 한다. 허브가 스위치와 다른 점은 연결된 기기 중 하나에서 전송된 패킷이 허브에 연결된 모든 기기로 브로드캐스팅된다는 점이다.  
  
반면 스위치는 패킷의 목적지 주소로 지정된 기기로 이어지는 포트로만 패킷이 전달된다. 또한 스위치는 일반적으로 LAN 세그먼트와 세그먼트에 연결된 허브를 연결한다. 스위치는 동일한 LAN 세그먼트의 기기를 향하는 트래픽을 걸러낸다. 이런 기능 덕분에 스위치는 자체 프로세싱 리소스와 네트워크 대역폭을 더 효율적으로 활용한다.

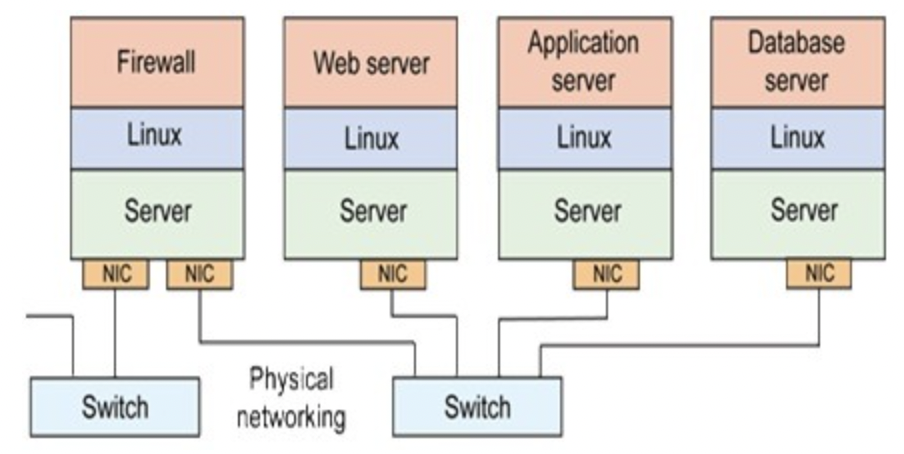
**6. 스위치와 라우터 비교**

스위치를 라우터와 혼동하는 경우도 종종 있다. 라우터는 네트워크 트래픽 포워딩과 라우팅 기능을 제공하며 라우터라는 이름도 여기에 유래한다. 그러나 이 작업을 수행하는 목적과 위치가 스위치와 다르다. 라우터는 계층 3(네트워크 계층)에서 작동하며 네트워크를 다른 네트워크로 연결하는 데 사용된다.  
  
스위치와 라우터 간의 차이를 쉽게 이해하려면 LAN과 WAN을 생각하면 된다. 기기는 스위치를 통해 로컬로 연결되고 네트워크는 라우터를 통해 다른 네트워크에 연결된다. 패킷이 인터넷에 이르기까지 일반적으로 거치는 경로(예를 들어 기기 > 허브 > 스위치 > 라우터 > 인터넷)를 생각해 보면 이해에 도움이 될 것이다.

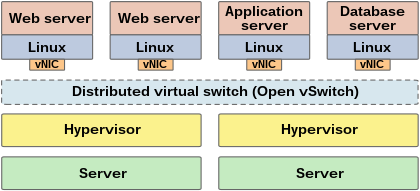
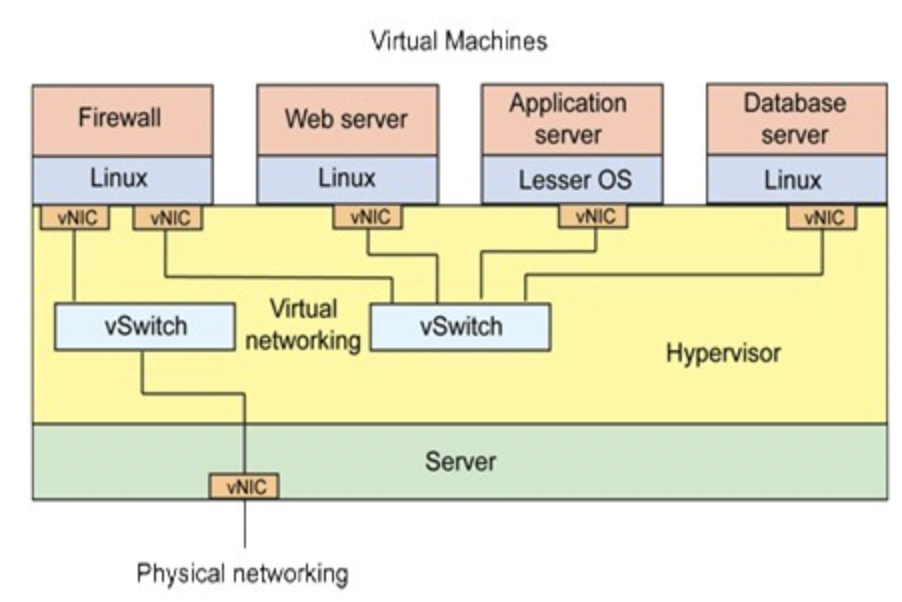
**7. Open VSwitch란?**

Apache 2.0 라이센스 기반의 오픈소스 가상 스위치로, 멀티 레이어 네트워크 스위치 기능을 하는 소프트웨어다. 가상화 기술로 하드웨어 스위치를 소프트웨어로 구현한다.

[기존의 스위치 구성도]



[가상화 스위치 구성도]



**8. DPDK**

DPDK는 Intel에서 개발한 고성능 패킷 처리 소프트웨어로 고속 패킷 처리를 위한 라이브러리와 드라이버를 제공하고 NFV의 네트워크 성능을 높이기 위한 핵심 기술이다. 리눅스 또는 윈도우의 Kernel 대신에 네트워크 패킷을 처리하는 응용 소프트웨어를 제공하고 전용 CPU core를 할당하여 네트워크 카드의 패킷을 Kernel을 거치지 않고 직접 처리한다.

기존 Kernel에서 네트워크 패킷을 처리하는 방식과는 다르게 응용 어플리케이션이 User Space의 DPDK 라이브러리를 사용하여 직접 엑세스한다는 점이 특징이다.

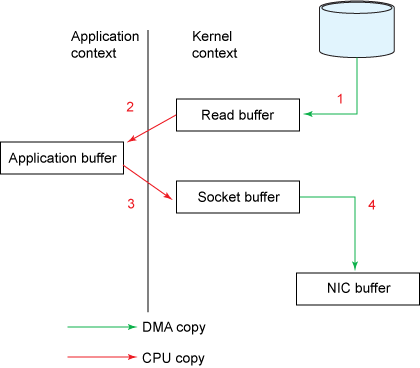
즉, 응용 어플리케이션을 특정 CPU core에 할당하여 가장 높은 우선 순위로 실행하고, Kernel을 거치지 않고 직접 네트워크 카드의 패킷을 처리하게 되면 네트워크 성능을 비약적으로 향상시킬 수 있다.

DPDK를 통해 해결할 수 있는 문제들은 아래 표와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**9. 일반적인 파일 전송과 Zero-Copy 방식 비교**

<일반적인 파일 전송>

일반적인 파일 ​전송시, 파일 서버나 정적 파일을 서비스하는 웹 애플리케이션은 디스크에서 파일 컨텐츠를 읽어 네트워크 소켓으로 데이터를 전송하는 일을 반복적으로 수행한다. 간단한 이 동작에는 운영체제 내부에서의 불필요한 컨텍스트 스위칭(Context Switching)과 데이터 복사(Data Copy)가 수반된다.

1) 유저가 read() 시스템 콜을 호출해 파일을 읽어달라고 요청하면, DMA 엔진에 의해서 디스크에 존재하는 파일의 내용이 커널 주소공간에 위치한 Read buffer에 복사된다.

2) 커널 주소공간에 위치한 Read buffer는 사용자가 접근할 수 없기 때문에 사용자가 read() 함수 호출시 파라미터로 전달한 Application buffer에 Read buffer의 내용을 복사해준다. 복사가 완료되면 호출된 함수가 반환된다.

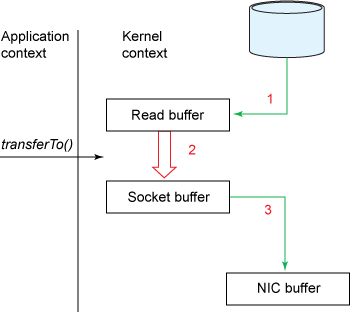
3) 사용자는 Application 버퍼로 읽어들인 데이터를 소켓으로 전송하기 위해 send() 함수를 호출한다. send() 함수 호출시 Application buffer를 파라미터로 전달하면 커널 영역에 위치한 Socket buffer로 데이터를 복사한다.

4) Socket buffer에 있는 데이터는 실제 네트워크로 전송하기 위해 네트워크 장비(NIC)에 있는 버퍼로 다시 복사된다.

4개의 버퍼에 동일한 데이터가 복사되는 이 과정은 애플리케이션의 CPU 자원을 소모한다. 이런 버퍼링은 본래 메모리와 디스크, 메모리와 네트워크 장비 사이의 속도 차이를 만회하고자 만들어진 성능 개선 장치다. 예를 들어 1KB 정도의 작은 데이터를 파일에서 읽으면 커널은 1KB 이후 데이터까지 한번에 읽어서 페이지 캐시에 로드한다. 이 후, 그 다음 데이터를 요청하면 물리적인 I/O 를 수행하지 않고 페이지 캐시에서 읽어서 사용자에게 준다. 데이터를 미리 읽어서 (Read Ahead) 성능 개선을 도모한 것이다. 그러나 이런 버퍼링이 성능 저하를 유발하는 병목(bottleneck)으로 작용할 수 있다. 사용자가 요청하는 데이터의 크기가 커널이 유지하는 버퍼의 사이즈보다 큰 경우 미리 읽는(Read Ahead) 성능 개선 효과보다 여러 단계에 걸쳐 데이터를 복사하는 비효율이 더 커지게 된다.

또한, 시스템 콜을 수행하기 위해 유저모드와 커널모드를 오가는 컨텍스트 스위칭(Context Switching)이 발생한다. 컨텍스트 스위칭은 수행 정보들을 백업하는 등의 오버헤드(Overhead)를 수반한다. 파일에서 읽어 소켓으로 전송하는 동작은 read() 함수의 호출과 반환, 소켓으로 데이터를 전송하는 send() 함수의 호출과 반환 등 총 4번의 컨텍스트 스위칭을 발생시킨다.

제로카피(Zero-Copy)는 이런 비효율적인 동작을 개선하기 위해 소개된 기법이다.

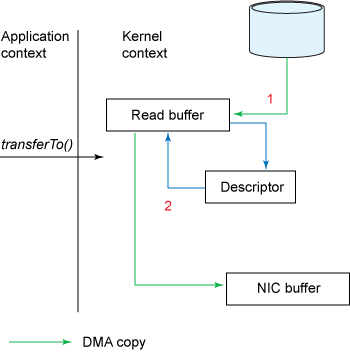
 <제로 카피>

1) 사용자가 transferTo() 메소드를 이용해 파일 전송을 요청한다. read()와 send() 함수가 하나로 합쳐진 형태의 시스템 콜이다. read() 시스템 콜과 마찬가지로 DMA 엔진이 디스크에서 파일을 읽어 커널 주소 공간에 위치한 Read buffer로 데이터를 복사한다.

2) 커널 모드에서 유저 모드로 컨텍스트 스위칭하지 않고 바로 Socket buffer로 데이터를 복사한다.

3) Socket buffer에 복사된 데이터를 DMA 엔진을 통해 NIC buffer로 복사되어 진다.

4) 데이터가 전송되고 transferTo() 메소드에서 리턴한다.

<개선된 제로 카피>

1) 사용자가 transferTo() 메소드를 호출한다. DMA 엔진이 디스크에서 파일을 읽어 커널에 위치한 Read buffer로 데이터를 복사한다.

2) 데이터가 소켓 버퍼로 복사되어지지는 않는다. 대신 데이터가 저장된 위치와 데이터 사이즈에 대한 정보와 함께 디스크립터(descriptor)가 소켓 버퍼에 추가된다. DMA 엔진은 이 정보를 이용해 Read buffer에 있는 데이터를 NIC 버퍼에 바로 복사하고, 네트워크로 데이터를 전송한다.