DPDK

Enrique Fernández Sánchez

Revisión 10 Marzo 2021

Índice

1 DPDK

1.1 ¿Qué es?

DPDK (*Data Plane Development Kit*) es un proyecto Open Source controlado por la Fundación Linux. Dicho proyecto tiene por objetivo proveer unas librerías de "plano de datos" y controladores para comunicarse con los propios controladores de las interfaces de red, con la finalidad de descargar el procesamiento de los paquetes TCP/UDP desde el núcleo del sistema operativo a los diferentes procesos que se ejecutan en el espacio de usuario (aplicaciones). Con esta descarga, lo que conseguimos es liberar al procesador de ciertas tareas especificas, dando por resultado una mayor eficiencia informática y un mayor rendimiento de paquetes, ya que introducimos mejoras a las propias instrucciones que nos proporciona el kernel.

Data Plane Development Kit surge a una alternativa competitiva para el procesamiento de paquetes de red. Si bien es cierto que actualmente el procesado que hace Linux de los paquetes es bastante eficiente, con DPDK podemos llevar a incrementar muchísimo la velocidad de procesado. Esto es muy importante ya que de cara a tecnologías futuras, comunicaciones de fibra óptica o el 5G, es muy necesario que los servidores del backbone, o aplicaciones específicas de routers y switches, puedan responder a las grandes necesidades que se nos proponen.

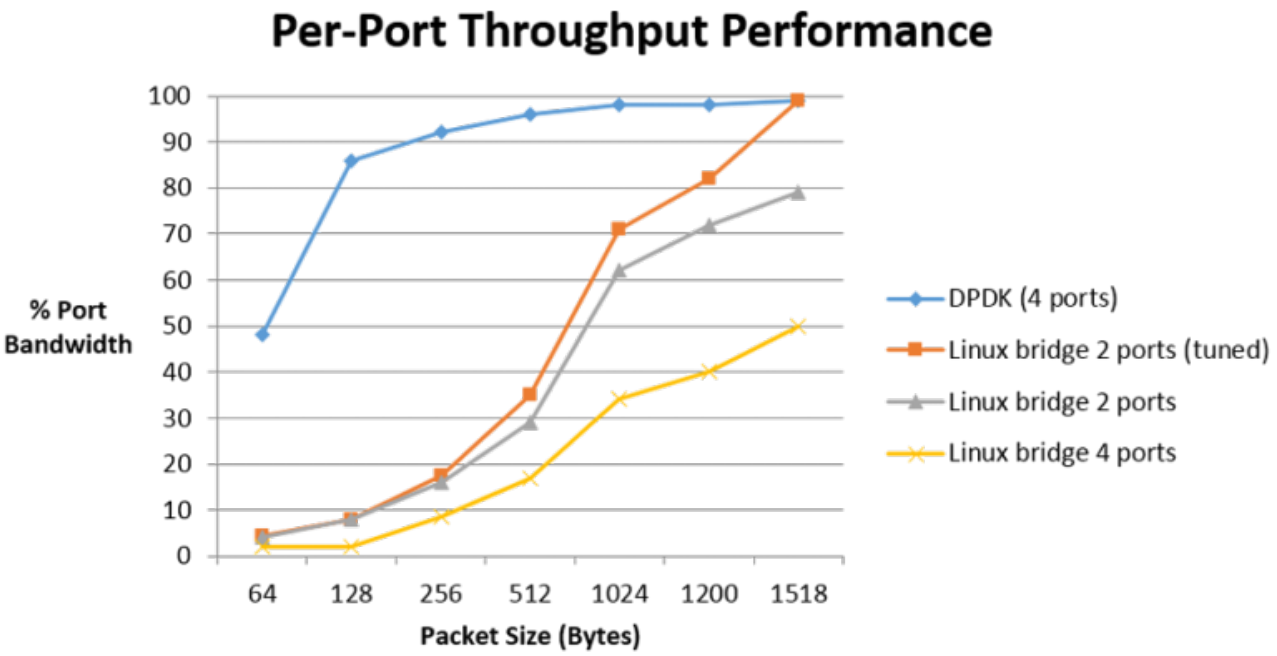

Figura 1: Comparación entre Linux y DPDK. (1)

1.2 Funcionamiento Lógico de *DPDK*.

La idea principal de *DPDK* es que una aplicación implementa dichas componentes y librerías que le permiten tomar el control completo de una tarjeta dered (por simplificar, en este caso es un único RJ45, pero podría ser cualquier interfaz de red o incluso más de una interfaz). Para esta aplicación *DPDK*, el sistema destina un *core* en especifico (*default* 1, pero se puede modificar para asignar más), además de la propia tarjeta de red. Estos recursos de los que la aplicación ha tomado el control, desaparecen del sistema para el usuario, y son dedicados íntegramente para *DPDK*.

Fijándonos directamente en la arquitectura de DPDK, tal y como vemos en (1), la comunicación entre la interfaz de red y la aplicación pasa de estar controlada por el Kernel, a que sea el propio DPDK el encargado de procesar y entender a la interfaz de red.

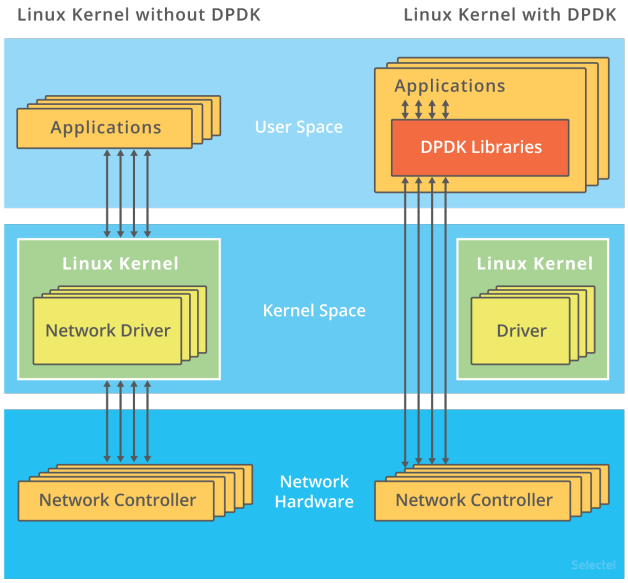

Figura 2: Comparación entre Linux y DPDK. (2)

Para que esto sea posible, hay otro componente que necesitamos, además de una interfaz de red compatible y una aplicación DPDK. Esto son las huge pages, suponiendo una parte crucial de la aplicación ya que nos permite alojar grandes porciones de datos y nos da la posibilidad de hacer rápidamente a ellos. En el caso de que los paquetes de red sean procesados por el kernel de Linux, utilizamos el acceso a memoría de tipo DMA (*Direct Memory Access*)

1.3 ¿Qué son las *huge pages*?

Para entender correctamente lo que son las *huge pages*, primero tenemos que profundizar en el concepto de *pages*. Cuando un proceso utiliza algo de memoria, la CPU marca la RAM como que está siendo utilizada por dicho proceso. Para aumentar la eficiencia, la CPU asigna la RAM en "porciones"(*chunks*) de 4K bytes (valor *default* en la mayoría de plataformas). Estas porciones se les llama *pages*.

Como el espacio de direccionamiento del proceso es virtual, la CPU y el SO tienen que recordar que páginas pertenecen a que procesos, y además, donde se almacenan. Por consecuencia, cuantas más páginas tengas, más tiempo tardarás en encontrar donde está la memoria alojada. En comparativa, si un proceso utiliza 1GB de memoria, haciendo la cuenta, serían 262144 entradas a comprobar (1GB/4K). Sin embargo, si una *Page Table Entry* utiliza 8 *bytes*, serían 262144 * 8 entradas a comprobar.

La mayoría de CPU actuales soportan páginas de mayor tamaño (por lo que la CPU tiene muchas menos entradas para comprobar), estas pueden recibir el nombre de *Huge pages* (en Linux), *Super pages* (en BSD) o *Larger pages* (en Windows), aunque todas son lo mismo.

1.3.1 Cómo configuramos las *huge pages* para DPDK.

Según la documentación oficial de *DPDK: Quick Start*, tenemos que realizar una intervención manual en la instalación de DPDK, en relación a las *huge pages*. Necesitamos reservar"las *huge pages*, lo haríamos tal que:

```
mkdir -p /dev/hugepages
mountpoint -q /dev/hugepages || mount -t hugetlbfs nodev /dev/hugepages
echo 64 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
```

1.4 Hardware soportado por DPDK.

El primer requisito de hardware para poder utilizar DPDK, es estar en disposición de una CPU compatible, aunque es lo común que así sea, es conveniente enumerarlas de cara a futuras aplicaciones. Dichas arquitecturas serían:

- **x86** (AMD, Intel)
- **PPC** (POWER9)
- **ARM** (BlueField, DPAA, DPAA2, OCTEON TX, OCTEON TX2)

En el sentido de las interfaces de red, tenemos mucha más variedad en donde elegir, tanto en lo que a fabricantes se refiere como en drivers de las propias tarjetas. Es importante tener en cuenta esta lista (y revisar adecuadamente la documentación de DPDK (*Supported Hardware*)) ya que dependiendo de la controladora que utilicemos, el driver puede tener funcionalidades específicas, además de cambiar los parámetros básicos que podemos entender como normales en dicha controladora (velocidad de transferencias, usar RJ-45 o utilizar fibra...). Siendo recomendable visitar la página oficial, aún podríamos listar las tarjetas de red compatibles con DPDK, siendo esa lista la siguiente:

- AMD
- Amazon
- Aquantia
- Atomic Rules
- Broadcom
- Chelsio
- Cisco
- Hisilicon
- Huawei
- Intel
- Marvell
- Mellanox
- NXP
- Netcope
- Netronome
- Solarflare
- Wangxun
- *Software NICs*

Como podemos comprobar, la lista de fabricantes que tienen NICs compatibles con DPDK es amplia. Por hacer más liviana la comprensión de los conceptos, podríamos destacar los siguientes fabricantes:

- Intel
- *Software NICs*

En el caso de Intel, es importante destacarle sobre todo ya que además de tener muchas drivers para una gran variedad de tarjetas de red, fue el precursor de DPDK. Al tener ya un rodaje, la mayoría de sus drivers tienen ya un recorrido por parte de la comunidad, situándolos como una opción muy competitiva respecto al resto de controladores.

Por otro lado, los *Software NICs* son importantes ya que, aunque no son un fabricante específico, nos permite utilizar un mínimo kernel de Linux para pasar la información de un controlador no soportado por DPDK, pero que tras estos controladores, si que puedan ser utilizados por aplicaciones DPDK.

Centrándonos de manera específica en estos dos fabricantes, podemos hacer una enumeración de drivers y de tarjetas soportadas.

- Intel (*Supported NICs Intel*)
  - e1000 [VM Emulated Devices]
  - e1000e
  - igb [1GbE]
  - igc [2.5GbE]
  - ixgbe [10G]
  - i40e [10/25/40G]
  - ice [10/25/50/100G]
  - fm10k [40/100GbE]
  - ipn3ke [FPGA Controller]
  - ifc
- Software NICs (*Supported Software NICs*)
  - AF_PACKET [AF_PACKET socket]
  - AF_XDP [AF_XDP socket]
  - tap/tun [kernel L2/L3]
  - pcap [file or kernel driver]
  - ring [memory]
  - memif [memory]

2 Glosario de términos.

- DPDK. *Data Plane Development Kit.*
- NICs. *Network Interface Card.*

3 Bibliografía

3.1 Links and references

1. Introduction to DPDK: Architecture and Principles
2. Wikipedia: Data Plane Development Kit
3. Hugepages
4. DPDK: Quick Start
5. Direct Memory Access (DMA)
6. Lista de arquitecturas e interfaces de red soportadas por DPDK.
7. NICs de Intel soportados por DPDK
8. Software NICs soportados por DPDK

3.2 Imágenes

1. Imagen comparativa de BW en diferentes tests case.
2. DPDK: How it works. General Features

A Anexos

A.1 Instalación de `ansible` para automatizar una VM

En el caso de que queramos utilizar la herramienta ansible para configurar una VM, tendremos que seguir los siguientes pasos.

1. Asegurarnos de que tenemos instalado el programa en el host. La principal dependencia de `ansible` es `Python`.
   Para instalar `ansible` en diferentes distribuciones sería tal que:

   - Ubuntu 21.04:

     Listing 1: Instalación `ansible` en Ubuntu

     ```
     sudo apt install -y software-properties-common
     sudo add-apt-repository --yes --update ppa:ansible/ansible
     sudo apt update
     sudo apt install -y ansible
     ```

   - Fedora 32:

     Listing 2: Instalación `ansible` en Fedora

     ```
     sudo dnf update
     sudo dnf install ansible
     ```

   - OpenSUSE:

     Listing 3: Instalación `ansible` en OpenSUSE

     ```
     zypper in ansible
     ```

   - Arch Linux:

     Listing 4: Instalación `ansible` en Arch Linux

     ```
     sudo pacman -S ansible
     ```

2. Creamos una clave SSH para utilizarla como método de autentificación con la maquina virtual.

   Listing 5: Crear clave SSH para autenticación en VM

   ```
   (host)$ ssh-keygen -t ed25519 -C "Host Ansible ssh key"
   ```

3. Utilizando nuestro gestor de máquinas virtuales, encendemos la VM. Es importante que nos aseguremos que tiene internet, por lo que configuramos que la interfaz de red sea de tipo *NAT*. Dentro de la máquina, buscaremos que ip tiene asignada con el comando `ip address`. Desde este momento, dejaremos la máquina virtual encendida.

4. Procedemos a copiar la clave pública SSH que hemos generado en el paso dos. Utilizamos el siguiente comando, sustituyendo con el usuario e IP específico de la máquina. Tendremos que configurar una red tipo *Host-only network* para poder comunicarnos correctamente con nuestra máquina virtual, para ello podemos seguir los pasos detallados en [**??**]

   Listing 6: Copiar clave pública de `ansible` a VM

   ```
   (host)$ ssh-copy-id -i $HOME/.ssh/id_ed25519.pub <user>@<IP VM>
   ```

5. En este momento ya podríamos conectar con la máquina virtual utilizando `ansible`. Sin embargo, por comodidad, lo que vamos a hacer es crear un archivo que funcione con inventario de servidores. Nos servirá para guardar las direcciones IP de los servidores en los que queremos ejecutar comandos remotos con `ansible`. Para ello, creamos un archivo `inventory`. Como ejemplo, dicho archivo puede ser tal que:

   Listing 7: Contenido del archivo `inventory` de `ansible`

   ```
   ## VMs locals
   [virtualbox]
   10.0.100.1
   10.0.100.2

   ## Server SSH
   [server]
   192.168.1.200
   ```

6. Para probar la conectividad. Nos aseguramos de que tenemos la máquina virtual encendida y que además, hemos copiado la clave ssh y tenemos su IP añadida a nuestro inventario. Después, procedemos a ejecutar el siguiente comando en la máquina host:

   ```
   (host)$ ansible -i inventory -m ping
   ```

   En la consola nos aparecerá la información de cada uno de las IP que habrá probado para ejecutar el comando remoto `ping`.

A.2 Configuración de *Guest Network* para comunicar con la VM

Para comunicar nuestra máquina host [**??**], con una *virtual machine* es necesario que estén en una misma red, es por esto por lo que vamos a crear una red en especifico para ello. En el programa *Virtualbox*, esta funcionalidad recibe el nombre de *Host-only*.

Es importante que cuando queramos configurar una máquina virtual con esta funcionalidad que siempre pongamos el adaptador de red 1 como el que tendrá la comunicación *guest*, mientras que el adaptador 2 será el que tendrá la conexión a internet via *NAT*.

Procedemos a enumerar los pasos a seguir para configurar una máquina virtual con conectividad con el host, utilizando *Virtualbox*.

1. Abrimos el programa *Virtualbox*. Navegamos a la sección: **File** → **Preferences**.

2. Seleccionamos la tabla de ***Host-only Networks***. Procedemos a pulsar el botón + para añadir una nueva red.

3. Asignamos las diferentes IP que considermos. Es importante que dejemos activado el servidor DHCP, así la máquina virtual tendrá una IP válida dentro de la red.

4. Ahora, lo que tenemos que hacer es añadir una interfaz de red a nuestra máquina virtual. Para ello, vamos a las **Setting**s de la máquina virtual. En el apartado de Network, cambiamos el **Adapter 1** a **Host-only Adapter**, y asignamos una segunda interfaz de red con NAT (Adapter 2 attached to NAT).

5. Para comprobar que lo hemos hecho correctamente. Arrancamos la máquina virtual, ejecutamos el comando `ip address`. Nos deberían aparacer las diferentes interfaces que previamente hemos configurado, es decir, una interfaz de loopback, otra que corresponde al Host-only network, y por último, una que corresponda con la interfaz de NAT.

A.3 Playbooks `ansible`