Projet Déploiement Réseau Temps Réel

Mise en œuvre d'un réseau AFDX à l'aide de la librairie DPDK

3SN parcours SEMBIIOT

2019-2020

1 Objectifs

Le principal objectif du projet est de construire une architecture réseau AFDX. Malheureusement, nous ne pouvons pas disposer de commutateur AFDX ni de calculateurs avion (End Systems). Aussi nous proposons d'utiliser la librairie DPDK afin de simuler le comportement d'un commutateur AFDX. Une fois compilé, le programme exécuté sur une station Linux jouera le rôle du commutateur. Le comportement des ES sera d'émettre des données de façon périodique pour simuler l'émission de données à chaque BAG.

2 La librairie DPDK

DPDK signifie Data Plane Developement Kit. C'est un ensemble de librairies et de pilotes de contrôleurs de cartes d'interfaces réseaux qui permettent d'améliorer la performance des traitements de paquets sur différents types de processeurs. Il est conçu pour fonctionner sur n'importe quel processeur.

DPDK est une solution logicielle de plan de données optimisée développée par Intel pour ses processeurs multi-coeurs. Cette librairie comprend un logiciel de traitement de paquets haute performance qui combine le traitement des applications, le traitement du contrôle, le traitement du plan de données et les tâches de traitement du signal sur une seule plate-forme. DPDK a une couche de bas niveau pour améliorer les performances. Elle possède des fonctions de gestion de la mémoire, un support d'interface réseau et des bibliothèques pour les classifications de paquets.

DPDK permet de faire abstraction du noyau Linux comme le montre la figure 1. Grâce aux librairies et aux pilotes, le contrôleur réseau communique directement avec l'application sans passer par le noyau. Cela permet de réduire fortement le temps de traitement d'un paquet. Autrement dit, DPDK permet de créer des applications qui manipulent les paquets de données en suivant le chemin le plus rapide au lieu de l'itinéraire normal des couches réseau et de la commutation contextuelle. Le matériel accéléré DPDK permet aux paquets de contourner entièrement le noyau.

3 Quelques caractéristiques AFDX

La transmission des données sur un réseau AFDX s'effectue à travers des Virtual Links (VL). Ces VLs sont des flux mono-émetteur et multicast.

La figure 2 montre les différents champs de la trame AFDX (qui est similaire à une trame Ethernet classique). Le champ adresse destination indique le numéro de VL. Les End-Systems émettent les

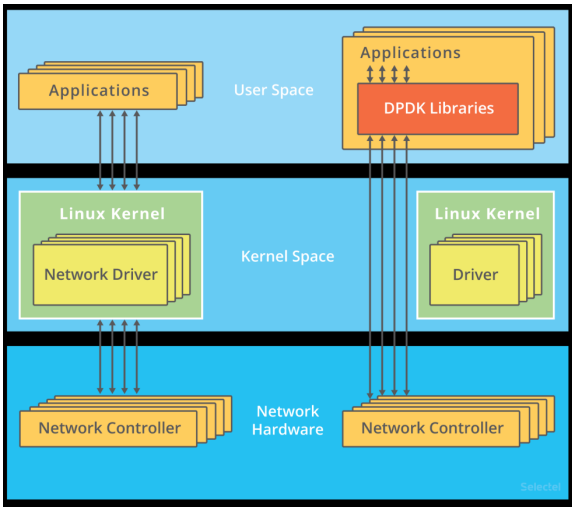

Figure 1 – Fonctionnement de DPDK : sans DPDK = passage par le noyau, avec DPDK = accès direct aux cartes réseau

trames AFDX régulièrement en respectant la contrainte du BAG (délai entre 2 trames d'un même VL, de valeur comprise entre 1 et 128 ms). Le rôle du commutateur est de transmettre les VLs arrivant d'un port d'entrée vers un ou plusieurs ports de sortie. Le filtrage des données dans le commutateur se fait au niveau de l'adresse MAC destination. La table de commutation doit donc reconnaître le numéro de VL et redirige la trame vers le ou les ports de sortie. Les trames sont émises sur les ports de sortie en respectant une politique FIFO (Aucune régulation n'est faite). Sur les équipements de bordure, la régularité du BAG est vérifiée sur les ports d'entrée.

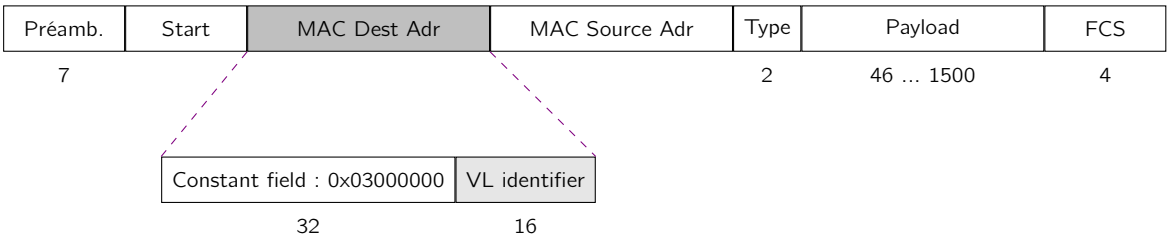

Figure 2 – Format d'une trame AFDX

4 Travail à faire

La librairie DPDK et l'ensemble des outils qu'elle utilise sont stockés dans le répertoire Reseau_TR.

4.1 Initialisation de l'environnement de DPDK

Avant de pouvoir utiliser DPDK, il faut construire les modules (pilotes Linux) pour gérer les cartes réseaux. Les buffers de transmission seront stockés en mémoire mais nécessitent une taille de page plus grande que celles utilisée par les systèmes Linux traditionnels. Aussi nous allons utiliser des « hugepages ». Pour configurer ces paramètres, un script shell `dpdk-setup.sh` est disponible dans le répertoire `usertools` dans le répertoire de DPDK. L'affichage de ce script ressemble à :

```
----------------------------------------------------------
 Step 1: Select the DPDK environment to build
----------------------------------------------------------
[1] arm64-armv8a-linuxapp-gcc
[2] arm64-dpaa2-linuxapp-gcc
[3] arm64-thunderx-linuxapp-gcc
[4] arm64-xgene1-linuxapp-gcc
[5] arm-armv7a-linuxapp-gcc
[6] i686-native-linuxapp-gcc
[7] i686-native-linuxapp-icc
[8] ppc_64-power8-linuxapp-gcc
[9] x86_64-native-bsdapp-clang
[10] x86_64-native-bsdapp-gcc
[11] x86_64-native-linuxapp-clang
[12] x86_64-native-linuxapp-gcc
[13] x86_64-native-linuxapp-icc
[14] x86_x32-native-linuxapp-gcc

----------------------------------------------------------
 Step 2: Setup linuxapp environment
----------------------------------------------------------
[15] Insert IGB UIO module
[16] Insert VFIO module
[17] Insert KNI module
[18] Setup hugepage mappings for non-NUMA systems
[19] Setup hugepage mappings for NUMA systems
[20] Display current Ethernet/Crypto device settings
[21] Bind Ethernet/Crypto device to IGB UIO module
[22] Bind Ethernet/Crypto device to VFIO module
[23] Setup VFIO permissions

----------------------------------------------------------
 Step 3: Run test application for linuxapp environment
----------------------------------------------------------
[24] Run test application ($RTE_TARGET/app/test)
[25] Run testpmd application in interactive mode ($RTE_TARGET/app/testpmd)

----------------------------------------------------------
 Step 4: Other tools
----------------------------------------------------------
[26] List hugepage info from /proc/meminfo

----------------------------------------------------------
 Step 5: Uninstall and system cleanup
----------------------------------------------------------
[27] Unbind devices from IGB UIO or VFIO driver
[28] Remove IGB UIO module
[29] Remove VFIO module
[30] Remove KNI module
[31] Remove hugepage mappings
```

Dans notre cas, le processeur est un `x86_64`. Il est possible de gérer 1024 « hugepages » de 2Mb.

1ère étape. Configurez l'environnement de DPDK.

4.2 Un premier essai

1. Récupérez les fichiers sur Moodle.
2. Compilez le fichier `l2fwd.c` à l'aide du Makefile.
3. Testez la transmission.

```
./build/l2fwd -- -p 0x3
```

4.3 Mise en œuvre de la commutation AFDX

Modifiez le code de l'application précédente pour simuler un commutation de type AFDX. Cette modification doit permettre de :

1. lire la configuration de la table de commutation ;
2. filtrer les VL AFDX et envoyer ces VLs vers le bon port de sortie.

Testez l'envoie de trames AFDX depuis une station de travail.

4.4 Les VL sont multicasts

Modifiez le code précédent pour simuler les VLs multicast.

4.5 Régulation des VL en entrée

Le but de cette dernière partie est de construire le comportement d'un End System. Il s'agit de transmettre les données en respectant les contraintes de BAG.