I. INTRODUCTION

II. CONSOMMATION DES NŒUDS

Dans un réseau informatique, la consommation des nœuds est un sujet très important à ne pas négliger lors de la mise sur pied de ce dernier.

I. 1- Définition

Entendons par **nœud** un point de connexion dans un réseau capable d'envoyer, de recevoir, de transmettre ou de stocker des données. Il s'agit ainsi des dispositifs tels que : un ordinateur, un routeur, un commutateur, un serveur ou tout autre appareil pouvant réaliser au moins l'une des fonctionnalités précédemment citées. Ainsi, pour assurer l'effectivité de toutes ces fonctions, il est nécessaire d'évaluer les besoins de chacun des nœuds pour une meilleure appréhension des sollicitations du réseau dans sa globalité : C'est dans ce contexte qu'on parlera de ***consommation des nœuds***.

II. 2- Rôles

L'étude de la consommation des nœuds va permettre d'identifier les besoins du réseau de part en part afin d'envisager des solutions pour une meilleure gestion des ressources disponibles, des prévisions sur les besoins, et la longévité. Ceci dans l'optique de garantir que le réseau en entier puisse satisfaire la demande des utilisateurs.

III. 3- Types de consommation

On distinguera plusieurs critères sur lesquels s'appuyer pour évaluer la consommation d'un réseau à savoir :

- **Consommation de la bande passante :** quantité de données transmise ou reçue par un nœud pendant une période donnée. Certains équipements en fonction de leur activités peuvent consommer une quantité considérable de la bande passante. En l'occurrence, un serveur de Streaming (netflix, youtube, spotify) transmet en continu de grandes quantités de données à de nombreux utilisateurs.
- **Consommation des ressources systèmes :** certains nœuds tels que les serveurs ou ordinateurs hébergeant des applications pourront consommer une quantité importante de ressources système comme la *puissance de calcul*, la *mémoire*, le *stockage*, etc. Par exemple, un serveur exécutant des bases de données complexes ou des algorithmes d'apprentissage automatique consommera beaucoup de ressources système.
- **Consommation d'énergie :** certains équipements réseaux tels que les commutateurs, les routeurs et serveurs peuvent consommer beaucoup d'énergie, ce qui peut devenir important à prendre en compte en terme de coût opérationnel et de durabilité environnementale. Cet aspect est souvent négligé lors de l'évaluation d'un réseau.

IV. 4- Application

La consommation des nœuds initialement définie prend tout son sens lorsque sa simulation s'applique dans différents domaines et cas de figures à l'instar de:

- Étude de l'impact environnemental des réseaux
- Gestion des ressources
- Choix plus optimal des équipements, de leur disposition et de leur utilisation
- Évaluation des performances du réseau

V. 5- Surveillance de la consommation des nœuds avec Ryu: SimpleMonitor

Il s'agit de l'outil principal de surveillance de la consommation des nœuds d'un réseau de Ryu. En effet, c'est le module de Ryu qui permet de suivre et de surveiller les statistiques (des commutateurs openflow) sur la consommation de la bande passante et

des performances des nœuds, le nombre de paquets reçus/transmis, la charge des ports, la génération des rapports sur les schémas de trafic etc. dans un réseau. Il permettra entre autres la détection des goulets d'étranglement (en surveillant la charge des nœuds et des liens, les problèmes de performances), la surveillance en temps réel.

L'utilisation de SimpleMonitor se fera généralement par écriture des scripts et des règles personnalisées pour la collecte et l'analyse des statistiques. On peut également le combiner avec d'autres modules de Ryu (comme ryu.lib.packet, ryu.lib.ofctl_v10_0 à ryu.lib.packet, ryu.lib.ofctl_v1_4, ryu.lib.dpid) pour obtenir des analyses plus poussées.

Vous verez plus bas un script permettant d'obtenir et d'afficher des statistiques du trafic :

```
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor13(simple_switch_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

Nous voyons ci-dessus un bout de code d'une fonction de surveillance du trafic qui a été implémentée par la classe SimpleMonitor13 du module du même nom. Dans cette classe, pendant le fonctionnement du switch dont on fat analyse, on crée un thread qui demandera de façon périodique (ici, toutes les 10 secondes) les informations statistiques sur le nœud concerné (`_monitor()`). D'autres threads seront crées cette fois ci pour détecter des évènements comme les changements d'état : connecté ou déconnecté (`EventOFPStateChange`), la cible du monitoring avec les mots clés `MAIN_DISPATCHER et DEAD_DISPATCHER.`

```
        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         '
                         'in-port  eth-dst          '
                         'out-port packets  bytes')
        self.logger.info('---------------- '
                         '-------- ----------------- '
                         '-------- -------- --------')
        for stat in sorted([flow for flow in body if flow.priority == 1],
                           key=lambda flow: (flow.match['in_port'],
                                             flow.match['eth_dst'])):
            self.logger.info('%016x %8x %17s %8x %8d %8d',
                             ev.msg.datapath.id,
                             stat.match['in_port'], stat.match['eth_dst'],
                             stat.instructions[0].actions[0].port,
                             stat.packet_count, stat.byte_count)

    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         port     '
                         'rx-pkts  rx-bytes rx-error '
                         'tx-pkts  tx-bytes tx-error')
        self.logger.info('---------------- -------- '
                         '-------- -------- -------- '
                         '-------- -------- --------')
        for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

Pour obtenir des réponses venant du nœud en l'occurrence un switch openflow, on fera appel à une classe `OPFFlowStatsReply` qui nous donnera une liste d'informations statistiques et les stockera pour chaque flux d'entrées soumis à une FlowStatsRequest. Par ailleurs, on peut également obtenir des réponses aux requêtes en créant un gestionnaire d'évènement pour recevoir les informations sur les ports : PortStatsReply message.

Une fois la rédaction de notre script achevée, on pourra lancer son exécution depuis notre contrôleur Ryu. Dans un premier temps, on observera des données nulles puisqu'il n'y a aucun flux entrée ; mais exécutant un ping entre deux hotes du réseaux, on pourra observer des données statistiques échangées.

Let's execute ping from host 1 to host 2.

host: h1:

```
# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=94.4 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 94.489/94.489/94.489/0.000 ms
#
```

Packet transfer and flow entry are registered and statistical information is changed.

controller: c0:

```
datapath           in-port  eth-dst           out-port packets  bytes
------------------ -------- ----------------- -------- -------- --------
0000000000000001         1 00:00:00:00:00:02        2        1       42
0000000000000001         2 00:00:00:00:00:01        1        2      140
datapath           port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
------------------ -------- -------- -------- -------- -------- -------- --------
0000000000000001         1        3      182        0        3      182        0
0000000000000001         2        3      182        0        3      182        0
0000000000000001         3        0        0        0        1       42        0
0000000000000001 fffffffe        0        0        0        1       42        0
```

III. MÉTHODE DE CLASSIFICATION DU TRAFIC

I. 1- naive

II. 2- machine learning

III. 3- Deep Learning

Il s’agit d’une autre application de l’intelligence artificielle toute aussi intéressante que la précédente qui consiste à utiliser les modèles de réseaux neuronaux profonds pour analyser, classifier et comprendre les schémas de trafic réseau. Nous verrons plus bas quelques aspects clés de cette approche.

- Rôle
  L’approche par Deep Learning joue un rôle déterminant dans la compréhension, l’optimisation et la sécurisation des réseaux. Ses algorithmes permettent de traiter une grande quantité de données de trafic et d’en extraire des informations significatives. Ainsi, on aura des prises de décisions plus intelligentes, une détection plus précise des schémas de trafic anormal ou malveillant du réseau et une amélioration globale de la qualité et de la fiabilité de ce dernier.

- Types d’algorithmes et domaines d’application
  On distinguera plusieurs types algorithme permettant d’implémenter cette approche :
  - Les réseaux neuronaux convolutifs (CNN) : généralement utilisés pour la classification d’images, ils sont capable d’extraire des caractéristiques importantes des données en entrées et sont grandement utilisés pour la détection des schémas de trafic.
  - Les réseaux neuronaux récursifs (RNN) et les cellules LSTM (Long Short-Terrm Memory) : Ils servent à comprendre des séquences de données et peuvent donc être utilisés pour comprendre les modèles temporels dans le trafic.
  - Les réseaux neuronaux profonds (RNN) : Ils sont utilisés pour des apprentissages automatiques plus complexes et des taches de classification avancées. Parmi ceux-ci on peut citer des réseaux neuronaux multicouches (MLP) et d’autres architectures plus évoluées.

- Domaines d’application
  Cette approche vaut son pesant d’or dans différents domaines d’application comme :
  - **La détection d’intrusion et sécurité réseau :** Ici, les techniques de Deep Learning permettront d’identifier les schémas de trafic malveillant ou non autorisé.
  - **La gestion de la bande passante et de la qualité du service :** On arrive à prédire la congestion du trafic et à optimiser la gestion de la bande passante et de la qualité de service dans le réseau.
  - **L’analyse des performances du réseaux :** L’analyse des schémas de trafic peuvent par cette approche aider à évaluer les performances du réseau, à anticiper les goulots d’étranglement et à optimiser les infrastructures réseau.

- Exemples
  Un exemple concret d’application de l’approche par Deep Learning pour la classification du trafic dans un réseau peut être la détection d’attaques par déni de service distribué (**DDos**) à l’aide de modèles de classification entraînés pour reconnaître les comportements anormaux dans le réseau.

Au vu de tout ce qui a été dit plus haut, on se rend compte avec aisance que lorsqu’on s’attaque à un réseau d’un certain niveau de complexité, il est important de faire appel à l’approche

par Deep Learning pour la classification du trafic afin d’avoir des données plus complètes et d’être mieux outillé pour fournir une meilleure qualité de service.