

Sommaire

1	Introduction	3
2	Rapport sur le Travaux pratiques	4
1	PRISE EN MAIN DE MININET-WIFI	4
2	DEFINITION DES TOPOLOGIES CUSTOMISEES	6
3	UTILISATION DE RYU DANS SON FONCTIONEMENT DE BASE	7
4	OPENFLOW ET OPENVSWITCH	7
3	Rapport sur l'analyse et la conception	14
1	Analyse des besoins	14
1.1	Exigences fonctionnelles	14
2	Exigences non fonctionnelles	14
3	Présentation des diagrammes	14
3.1	Diagramme de contexte	14
3.2	Diagramme de cas d'utilisation	15
3.3	Description textuelle des cas d'utilisations	15
3.4	Diagramme de sequence technique	19
3.5	Diagramme d'activités	21
4	Rapport sur la modélisation	22
1	Programmation linéaire en nombres entiers	22
2	Contexte	23
2.1	Architecture étudiée dans notre modélisation : réseau maillé sans fils	24
3	Modélisation EAR-RP	25
3.1	Vue d'ensemble d'EAR-RP	25
3.2	Modèle de conception EAR-RP	25
4	Programme linéaire en nombres entiers du EAR-RP	26
5	EAR-RP : évaluation et discussion	27
5.1	Architecture logique	27
5.2	Mesures de simulation et d'évaluation des performances	30
5.3	Résultats et discussion	30

5	Rapport sur l'implémentation	34
1	Mininet-Wifi	34
2	Etapes de la simulation	34
2.1	Création de la topologie	34
2.2	Lancement de la topologie	35
2.3	Recherche de tous les chemins existants entre deux nœuds	36
2.4	Recherche des chemins possibles selon l'EAR-RP	37
2.5	Rechercher le chemin optimal	39
2.6	Affichons les energies résiduelles des noeuds	41
6	Conclusion	43

Introduction

Ce document présente une approche efficace visant à minimiser la consommation totale d'énergie tout en tenant compte des contraintes de capacité d'énergie résiduelle. Résoudre le problème complexe du routage économique en énergie implique la recherche du chemin optimal pour acheminer chaque flux de la source à la destination, tout en optimisant la consommation d'énergie. Il est essentiel de définir précisément les règles et de les intégrer aux points d'accès, car cela peut avoir un impact significatif sur la consommation d'énergie. Notre principale préoccupation concerne les zones où l'alimentation électrique est limitée, insuffisante ou irrégulière. Dans de telles conditions, il est impératif que les nœuds du réseau fonctionnent de manière autonome sur le plan énergétique. Ainsi, il est essentiel de prendre en compte à la fois la consommation d'énergie totale et l'énergie résiduelle de chaque nœud lors du calcul du chemin du réseau. Ces mesures sont cruciales pour évaluer la durée de vie globale du réseau et des nœuds. Pour résoudre ce problème, nous formulons une approche basée sur un programme linéaire entier et proposons une solution heuristique gloutonne.

Ainsi, nous allons :

1. Proposer une nouvelle approche de routage conscient de l'énergie résiduelle des nœuds dans un réseau maillé ;
2. Modéliser l'approche sous forme d'un programme linéaire entier ;
3. Proposer une heuristique pour résoudre notre programme linéaire entier ;
4. Evaluer les performances de notre solution en termes d'énergie et de durée de vie du réseau pour voir si notre solution permet d'étendre simultanément la durée de vie du réseau et des nœuds par rapport à l'algorithme de Dijkstra, largement utilisé pour le routage mais ne tenant pas compte des contraintes d'énergie.

Rapport sur le Travaux pratiques

1 PRISE EN MAIN DE MININET-WIFI

Sudo mn & sudo mn -c wifi De quoi est composée la topologie par défaut de mininet (combien de switch open flow et combien d'hôtes) ?

La topologie est compose de 2 hôtes h1 h2 et un switch open flow S1

```
(base) bessala@bessala-VirtualBox:~$ sudo mn
[sudo] Mot de passe de bessala :
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> █
```

A quoi servent les commandes pingall, iperf, xterm, ifconfig, dump, ‘ links et net de mininet-wifi ?

1. pingall : Cette commande permet de tester la connectivité entre tous les hôtes de la topologie Mininet-WiFi. Elle envoie des paquets ICMP (Ping) entre tous les hôtes pour vérifier s'ils peuvent se joindre les uns aux autres.

2. iperf : La commande iperf est utilisée pour mesurer la bande passante entre les hôtes de la topologie Mininet-WiFi. Elle permet de générer du trafic réseau et de mesurer les performances du réseau, notamment la vitesse de transfert des données.

3. xterm : La commande xterm est utilisée pour ouvrir une fenêtre de terminal séparée pour un nœud spécifique de la topologie Mininet-WiFi. Cela permet d'exécuter des commandes spécifiques sur ce nœud et de voir la sortie dans la fenêtre de terminal séparée.

4. ifconfig : La commande ifconfig est utilisée pour afficher et configurer les interfaces réseau sur les hôtes de la topologie Mininet-WiFi. Elle permet de voir les adresses IP, les masques de sous-réseau et d'autres informations liées à l'interface réseau.

```
(base) root@bessala-VirtualBox:~# ifconfig
h1:eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
        ether 12:53:eb:dd:af:4e txqueuelen 1000 (Ethernet)
          RX packets 1154739 bytes 7623029 (7.6 MB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 1473739 bytes 6734507230 (6.7 GB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        ether ::1 txqueuelen 1000 (Boucle locale)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 0 bytes 0 (0.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(base) root@bessala-VirtualBox:~# []

mininet>
bessala@bessala-VirtualBox:~
Fichier Édition Affichage Rechercher Terminal Aide
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['10.8 Gbits/sec', '10.8 Gbits/sec']
mininet> xterm
usage: xterm node1 node2 ...
mininet> ifconfig
*** Unknown command: ifconfig
mininet> ipconfig
*** Unknown command: ipconfig
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3647>
<Host h2: h2-eth0:10.0.0.2 pid=3649>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3654>
<Controller c0: 127.0.0.1:6653 pid=3640>
mininet> xterm h1
mininet>
```

5. dump : La commande dump est utilisée pour afficher les informations détaillées sur la topologie Mininet-WiFi. Elle affiche les nœuds, les interfaces, les adresses IP, les tables de routage et d'autres informations utiles sur la topologie en cours d'exécution.

6. links : La commande links est utilisée pour afficher les connexions entre les nœuds dans la topologie Mininet-WiFi. Elle affiche les liens physiques et virtuels entre les nœuds, y compris les détails tels que les débits, les latences, les pertes de paquets, etc.

7. net : La commande net est utilisée pour afficher les informations globales sur la topologie Mininet-WiFi en cours d'exécution. Elle affiche les nœuds, les liens, les contrôleurs, les sous-réseaux, les interfaces et d'autres informations générales sur la topologie.

Les switchs de cette instance mininet sont configurés en Learning switch ? par défaut. Qu'est-ce qu'un Learning switch ?

Un "Learning switch" est un type de switch qui utilise la méthode d'apprentissage d'adresses MAC pour apprendre et mémoriser les adresses MAC des hôtes connectés à ses ports. Lorsqu'un switch reçoit un paquet réseau, il examine l'adresse MAC source du paquet et met à jour sa table d'apprentissage en associant cette adresse MAC à un port spécifique

Quel est le résultat de pingall lorsque le Learning switch est utilisé ?

Le pingall va marcher

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

Quittons mini net-wifi et redémarrez en désactivant le contrôleur (Controller none). Quel est le résultat du pingall ? Quel semble donc être le rôle du contrôleur dans une architecture SDN ?

Suite au pingall la communication entre h1 et h2 n'existe plus. Le pingall a échoué Le rôle du contrô-

```
(base) bessala@bessala-VirtualBox:~$ sudo mn --controller none
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
```

leur dans une architecture SDN (Software-Defined Networking) est de centraliser le contrôle et la Dans les switches L2 traditionnels, il existe une séparation entre le plan de contrôle et le plan de données.

2 DEFINITION DES TOPOLOGIES CUSTOMISEES

La topologie utilisée jusqu'ici correspond à : mn ?topo=tree, depth=1, fanout=2 Si l'on modifie maintenant cette topologie et que l'n crée une topologie d'une profondeur de 3 et d'un fanout de 4, combien y-a-t-il de switches ? Nous comptons Combien d'hôtes ? Combien de liens et enfin combien de contrôleurs A quel switch est relié l'ôte 25 ? Afficher cette topologie.

★ **Nous comptons 21 switchs et 64 hôtes**

★ **L'hôte 25 est liée au switch switch 10**

3 APIs sont essentielles à la définition d'une topologie : addSwitch, addHost et addLink.

Créer un fichier python dans lequel vous allez grâce à ces différentes fonctions créer une topologie qui correspondra à la topologie d'écrite ‘a la question 1.

```

Completed in 80.010 seconds
base) bessala@bessala-VirtualBox:~$ sudo mn --topo=tree,depth=3,fanout=4
** Creating network
** Adding controller
** Adding hosts:
1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h3
h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61
h62 h63 h64
** Adding switches:
1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21
** Adding links:
s1, s2) (s1, s7) (s1, s12) (s1, s17) (s2, s3) (s2, s4) (s2, s5) (s2, s6) (s3, h1) (s3, h2) (s3, h3) (s3, h4) (s4, h5
(s4, h6) (s4, h7) (s4, h8) (s5, h9) (s5, h10) (s5, h11) (s5, h12) (s6, h13) (s6, h14) (s6, h15) (s6, h16) (s7, s8)
s7, s9) (s7, s10) (s7, s11) (s8, h17) (s8, h18) (s8, h19) (s8, h20) (s9, h21) (s9, h22) (s9, h23) (s9, h24) (s10, h2
(s10, h26) (s10, h27) (s10, h28) (s11, h29) (s11, h30) (s11, h31) (s11, h32) (s12, s13) (s12, s14) (s12, s15) (s12
s16) (s13, h33) (s13, h34) (s13, h35) (s13, h36) (s14, h37) (s14, h38) (s14, h39) (s14, h40) (s15, h41) (s15, h42)
s15, h43) (s15, h44) (s16, h45) (s16, h46) (s16, h47) (s16, h48) (s17, s18) (s17, s19) (s17, s20) (s17, s21) (s18, h
9) (s18, h50) (s18, h51) (s18, h52) (s19, h53) (s19, h54) (s19, h55) (s19, h56) (s20, h57) (s20, h58) (s20, h59) (s2
, h60) (s21, h61) (s21, h62) (s21, h63) (s21, h64)
** Configuring hosts
1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h3
h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61
h62 h63 h64
** Starting controller
0
** Starting 21 switches
1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 ...
** Starting CLI:
ininet> net

```

3 UTILISATION DE RYU DANS SON FONCTIONEMENT DE BASE

Dans le terminal 1, ryu run ryu/app/simple switch 13.py

Dans le terminal 2 demarer une topologie customisée, sudo python myapp.py

Quel est le résultat d'un pingall ?

Grace à une commande vue précédemment, indiquez les liens entre les différentes interfaces (s1-eth1 :h1-eth0, etc.).

En modifiant votre fichier de topologie, supprimez le lien entre s1 et s2. Essayez à nouveau d'effectuer un pingall, que se passe-t-il ?

4 OPENFLOW ET OPENVSWITCH

Comme vous le savez, une architecture SDN est composée de trois couches principales : Application - Contrôle - Infrastructure.

Fonctionnement de switches traditionnels ; Rappelez le fonctionnement des switches L2 traditionnels : **Également appelés commutateurs Ethernet, fonctionnent au niveau de la couche 2 du modèle OSI (Open System Interconnections).**

-Existe-t-il une séparation entre le plan de contrôle et le plan des données ?

Dans les switches L2 traditionnels, il existe une séparation entre le plan de contrôle et le plan de données.

Le plan de contrôle est responsable de la gestion et de la configuration du switch, tandis que le plan de données est responsable du transfert des paquets

```
1 # -*- coding: utf-8 -*-
2
3 from mininet.net import Mininet
4 from mininet.topo import Topo
5
6 class TreeTopo(Topo):
7     def build(self, depth=1, fanout=2):
8         self.addTree(depth, fanout)
9
10    def addTree(self, depth, fanout):
11        if depth <= 0:
12            return
13
14        switch = self.addSwitch('s{}'.format(depth))
15
16        for _ in range(fanout):
17            host = self.addHost('h{}'.format(depth))
18            self.addLink(host, switch)
19
20        for _ in range(fanout):
21            self.addTree(depth - 1, fanout)
22
23 # Création de la topologie
24 topo = TreeTopo(depth=3, fanout=4)
25
26 # Initialisation du réseau Mininet
27 net = Mininet(topo)
28
29 # Démarrage du réseau
30 net.start()
```

-Quel type de données contient la ?Forwarding Table ?

La "Forwarding Table" (table de transfert) d'un switch L2 traditionnel contient des informations sur les adresses MAC et les ports associés. Elle indique comment les paquets doivent être acheminés en fonction des adresses MAC de destination

Quel type de données est traité au niveau 2 ?

Au niveau 2, les données traitées sont des trames Ethernet, qui incluent les adresses MAC source et de destination.

-Comment cette table est-elle mise à jour ?

La mise à jour de la table de transfert dans un switch L2 traditionnel se fait généralement par apprentissage automatique (MAC learning). Lorsqu'un switch reçoit une trame, il enregistre l'adresse MAC source et le port d'entrée associé dans sa table de transfert. Lorsqu'il doit transférer une trame vers une adresse MAC de destination inconnue, il diffuse la trame à tous les ports, à l'exception du port d'entrée. Lorsqu'il reçoit une réponse de la destination, il enregistre également l'adresse MAC de destination et le port associé dans sa table de transfert.

Switch SDN bases sur Open flow ; listez les principaux messages qu'Open Flow doit permettre d'échanger : Pensez à indiquer l'émetteur (contrôleur ou switch) et le destinataire (contrôleur ou switch) ainsi que leur raison d'être

- * **Message Packet-In :** Émis par le switch et envoyé au contrôleur pour les paquets qui ne peuvent pas être traités localement par le switch.
- * **Message Flow-Mod :** Émis par le contrôleur et envoyé au switch pour installer une règle de flux (flow rule) spécifiant comment traiter les paquets.
- * **Message Flow-Removed :** Émis par le switch et envoyé au contrôleur lorsqu'une règle de flux est supprimée.
- * **Message Get-Config :** Émis par le contrôleur pour récupérer la configuration du switch.
- * **Message Features-Reply :** Émis par le switch et envoyé au contrôleur pour fournir des informations sur les fonctionnalités du switch.
- * **Message Port-Status :** Émis par le switch et envoyé au contrôleur pour signaler les changements d'état des ports du switch.

Dans un terminal 1, Relancer un contrôleur Ryu avec un switch de niveau 2 : ryu-manager ryu/ryu/app/simple_switch_13.py

Dans un terminal 2, sudo mn ?controller=remote switch=ovsk, protocols=OpenFlow13 ?topo=linear, 6

Ce que nous allons faire est observer les échanges se produisant entre les différents switches, ainsi qu'entre les switches et le contrôleur. Pour ce faire nous allons lancer Wireshark et observer les échanges qui se produisent en local (interface loopback)

Lancez maintenant la commande pingall : Quel type de commandes Open Flow sont capturées par wireshark, d'après la partie théorique quelle est leur rôle ?

```

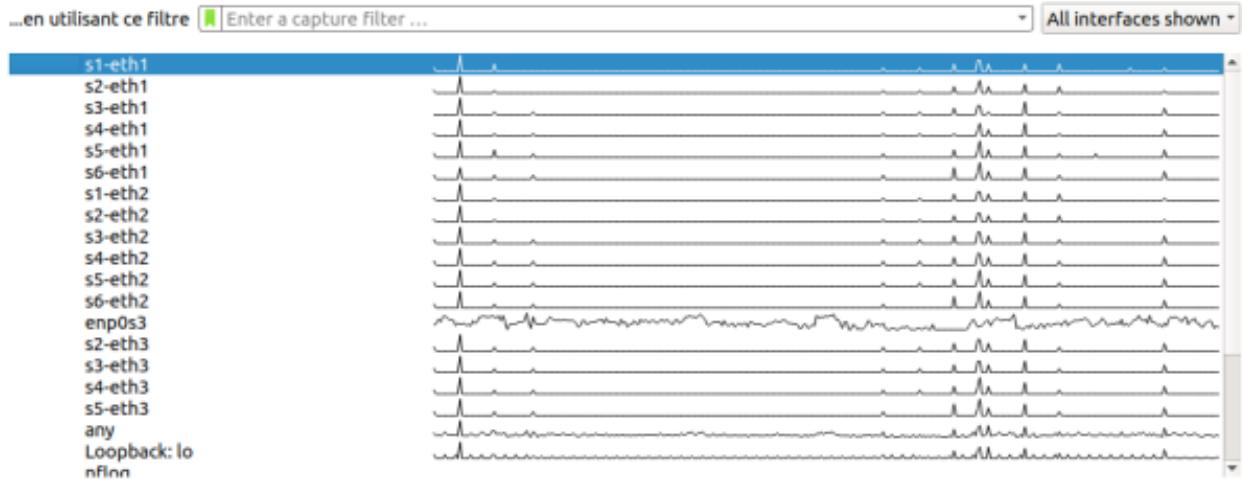
Fichier Édition Affichage Rechercher Terminal Aide
(base) bessala@bessala-VirtualBox:~$ ryu-manager ryu/ryu/app/simple_switch_1
3.py
Registered VCS backend: git
Registered VCS backend: hg
Registered VCS backend: svn
Registered VCS backend: bzr
loading app ryu/ryu/app/simple_switch_13.py
loading app ryu.controller.ofp_handler
instantiating app ryu/ryu/app/simple_switch_13.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 0000000000000001 62:a2:ad:3e:6c 33:33:00:00:00:16 1
packet in 0000000000000002 62:a2:ad:3e:6c 33:33:00:00:00:16 2
packet in 0000000000000005 c6:6b:51:8f:6d:57 33:33:ff:8f:6d:57 2
packet in 0000000000000006 c6:6b:51:8f:6d:57 33:33:ff:8f:6d:57 2
packet in 0000000000000005 62:de:b9:0e:28:0f 33:33:ff:0e:28:0f 1
packet in 0000000000000004 62:de:b9:0e:28:0f 33:33:ff:0e:28:0f 2
packet in 0000000000000006 62:de:b9:0e:28:0f 33:33:ff:0e:28:0f 2
packet in 0000000000000004 62:de:b9:0e:28:0f 33:33:ff:0e:28:0f 3
packet in 0000000000000005 62:a2:ad:3e:6c 33:33:00:00:00:16 2
packet in 0000000000000004 d6:66:1f:a2:a7:b5 33:33:ff:a2:a7:b5 3
packet in 0000000000000002 da:7d:79:b1:42:07 33:33:00:00:00:16 1
packet in 0000000000000003 62:de:b9:0e:28:0f 33:33:ff:0e:28:0f 3
packet in 0000000000000001 62:a2:ad:3e:6c 33:33:ff:3e:67:6c 1
SDNKitNet...

```

```

Fichier Édition Affichage Rechercher Terminal Aide
(base) bessala@bessala-VirtualBox:~$ sudo mn --controller=remote --switch protocols=OpenFlow13 --topo=linear,6
[sudo] Mot de passe de bessala :
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (h6, s6) (s2, s1) (s3, s2)
(s5, s4) (s6, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
mininet>

```



Après avoir lancé Wireshark, les commandes Open Flow capturées seront principalement les suivantes :

- * **OFPT_HELLO** : Cette commande est envoyée par les switches pour s'identifier auprès du contrôleur.
- * **OFPT\$FEATURES\$REQUEST et OFPT\$FEATURES\$REPLY** : Ces commandes sont utilisées pour échanger des informations sur les fonctionnalités des switches entre le contrôleur et les switches.
- * **OFPT\$PACKET\$IN** : Cette commande est envoyée par les switches au contrôleur pour lui transmettre des paquets entrants qui ne peuvent pas être traités localement.
- * **OFPT\$PACKET\$OUT** : Cette commande est envoyée par le contrôleur aux switches pour leur indiquer comment envoyer des paquets sortants vers une destination donnée.

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6
h2 -> h1 h3 h4 h5 h6
h3 -> h1 h2 h4 h5 h6
h4 -> h1 h2 h3 h5 h6
h5 -> h1 h2 h3 h4 h6
h6 -> h1 h2 h3 h4 h5
*** Results: 0% dropped (30/30 received)
mininet>

```



- * **OFPT\$FLOW\$MOD** : Cette commande est utilisée par le contrôleur pour installer, modifier ou supprimer des règles de flux (flow rules) dans les switches.

Si vous relancez ‘à nouveau la commande pingall, quelle différence observez-vous avec la question précédente ? Pourquoi ?

Nous remarquons que les commandes capturées par Wireshark seront différentes. En particulier, vous observerez les commandes

OFPT\$PACKET\$IN et OFPT\$PACKET\$OUT plus fréquemment. Cela s’explique par le fait que lors de l’exécution de la commande pingall, les switches envoient les paquets ICMP (Ping) au contrôleur via la commande OFPT\$PACKET\$IN, et le contrôleur renvoie ensuite les paquets de réponse aux switches via la commande OFPT\$PACKET\$OUT.

Les switches SDN fonctionnent de manière différente des switches traditionnels (legacy devices) qui fonctionnent sans SDN. Voici les principales différences entre les deux :

1. Fonctionnement des switches SDN :

- * Les switches SDN sont des switches programmables qui séparent le plan de contrôle (control plane) du plan de données (data plane).

- ★ Le plan de contrôle est géré par un contrôleur SDN qui prend les décisions de transfert des paquets et contrôle le comportement des switches.

- ★ Le plan de données, également appelé "forwarding plane", est responsable du transfert réel des paquets en fonction des instructions du contrôleur.

2. Différence avec les switches traditionnels :

- ★ Dans les switches traditionnels, le plan de contrôle et le plan de données sont combinés au sein du switch lui-même, et les décisions de transfert des paquets sont prises localement par le switch.

- ★ Les switches SDN offrent une flexibilité et une programmabilité accrues. Le contrôleur SDN peut prendre des décisions de transfert de manière centralisée, ce qui permet de mettre en œuvre des politiques de réseau plus complexes et d'adapter dynamiquement le comportement des switches en fonction des besoins du réseau.

3. Données traitées par le "forwarding plane" :

- ★ Le "forwarding plane" des switches SDN traite principalement des paquets de données.

- ★ Les commandes OFPT\$PACKET\$IN et OFPT\$PACKET\$OUT, qui sont capturées par Wireshark, sont utilisées pour transmettre des paquets au contrôleur (OFPT\$PACKET\$IN) et pour renvoyer des paquets du contrôleur aux switches (OFPT\$PACKET\$OUT).

4. Rôle du contrôleur :

- ★ Le contrôleur SDN joue un rôle central dans le réseau SDN. Il est responsable de la gestion, du contrôle et de la configuration des switches.

- ★ Le contrôleur communique avec les switches en utilisant le protocole OpenFlow pour installer, modifier et supprimer des règles de flux (flow rules) dans les switches.

- ★ Le contrôleur reçoit les paquets entrants des switches via les commandes OFPT\$PACKET\$IN et peut prendre des décisions de transfert en fonction de ces paquets.

- ★ Le contrôleur envoie également des paquets sortants aux switches via les commandes OFPT\$PACKET\$OUT pour contrôler le transfert des données.

Remarque : L'outil en ligne de commande ovs-ofctl peut être utilisé pour superviser et gérer les switches Open Switch du réseau SDN. Il permet d'interagir avec les switches en utilisant le protocole Open Flow et

d'effectuer des opérations telles que l'installation de règles de flux, la modification de la table de transfert.

Rapport sur l'analyse et la conception

1 Analyse des besoins

1.1 Exigences fonctionnelles

Il s'agit ici de l'ensemble des fonctionnalités qu'un système met à la disposition des utilisateurs. L'application devra proposer les fonctionnalités suivantes :

- Créer une station
- Supprimer une station
- Créer un access point
- Supprimer un accesspoint
- Ajouter un lien
- Supprimer un lien
- Afficher les chemins existants entre deux nœuds
- Afficher les chemins possibles entre deux nœuds vérifiant les critères de l'EAR-RP
- Afficher le chemin optimal selon l'EAR-RP
- Afficher le chemin optimal selon Dijkstra
- Router un paquet entre deux nœuds selon l'algorithme de Dijkstra
- Simuler l'activité du réseau avec l'EAR-RP
- Simuler l'activité du réseau avec l'algorithme de Dijkstra
- Afficher les énergies résiduelles des différents nœuds du réseau

2 Exigences non fonctionnelles

Les exigences non fonctionnelles sont l'ensemble des contraintes dans la réalisation des fonctionnalités d'un système. Nous avons à cet effet :

- La solution devra être sous forme d'application mobile

3 Présentation des diagrammes

3.1 Diagramme de contexte

Ce diagramme montre l'interaction entre l'utilisateur et l'application

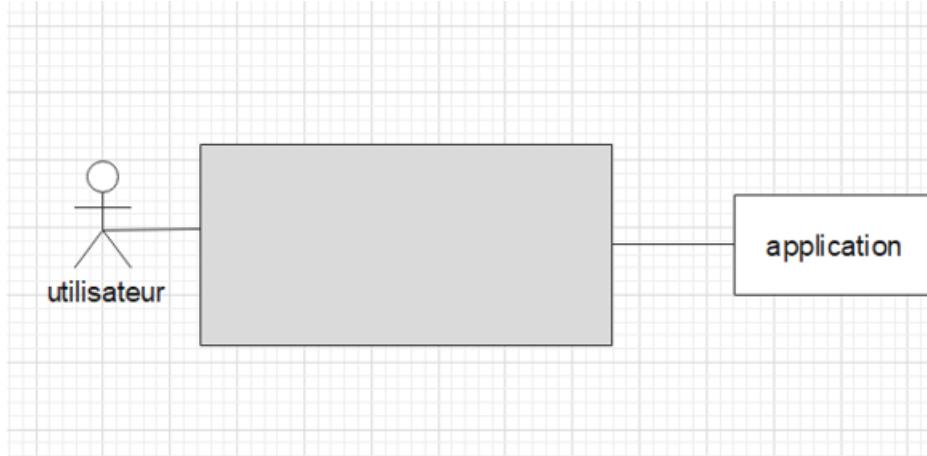


FIGURE 3.1 – Diagramme de contexte

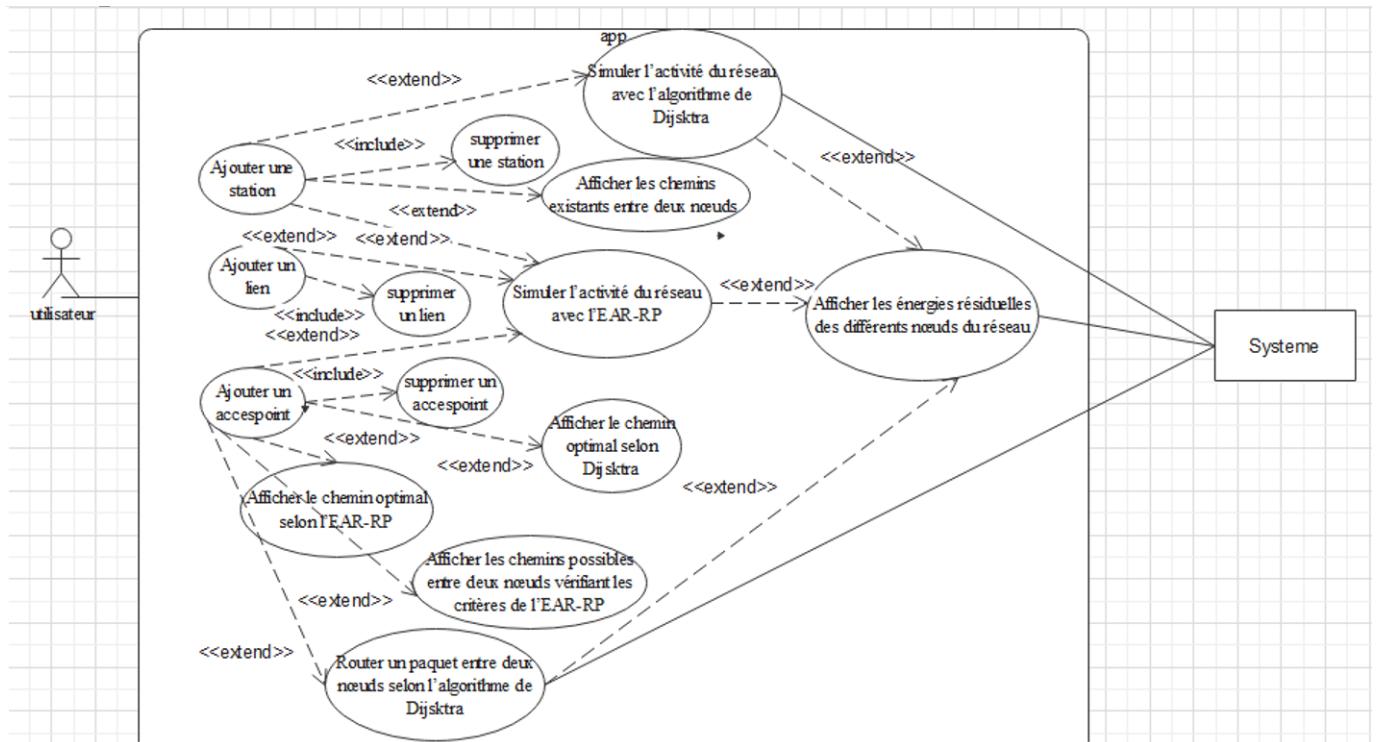


FIGURE 3.2 – Diagramme de cas d'utilisation

3.2 Diagramme de cas d'utilisation

3.3 Description textuelle des cas d'utilisations

Créer une station

- Description : Ce cas d'utilisation permet à un utilisateur de créer une nouvelle station dans le système. Une station peut représenter un dispositif ou un noeud dans un réseau.
- Acteurs : utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour créer une station.
- Postconditions : Une nouvelle station est créée dans le système avec les informations fournies par

l'utilisateur.

Supprimer une station

- Description : Ce cas d'utilisation permet à un utilisateur de supprimer une station existante dans le système.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour supprimer une station.
- Postconditions : La station spécifiée est supprimée du système, ainsi que tous les liens ou accès points associés.

Créer un point d'accès(access point)

- Description : Ce cas d'utilisation permet à un utilisateur de créer un nouvel accès point dans le système. Un accès point peut être un élément permettant la connectivité sans fil dans un réseau.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour créer un accès point.
- Postconditions : Un nouvel accès point est créé dans le système avec les informations fournies par l'utilisateur.

Supprimer un point d'accès(access point)

- Description : Ce cas d'utilisation permet à un utilisateur de supprimer un accès point existant dans le système.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour supprimer un accès point.
- Postconditions : L'accès point spécifié est supprimé du système, ainsi que tous les liens associés.

Ajouter un lien

- Description : Ce cas d'utilisation permet à un utilisateur d'ajouter un lien entre deux nœuds dans le système. Un lien représente une connexion ou une relation entre deux nœuds dans un réseau.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour ajouter un lien.
- Postconditions : Un nouveau lien est créé entre les deux nœuds spécifiés dans le système.

Supprimer un lien

- Description : Ce cas d'utilisation permet à un utilisateur de supprimer un lien existant entre deux nœuds dans le système.
- Acteurs : Utilisateur

-
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour supprimer un lien.
 - Postconditions : Le lien spécifié est supprimé du système.

Afficher les chemins existants entre deux noeuds

- Description : Ce cas d'utilisation permet à un utilisateur d'afficher tous les chemins existants entre deux noeuds dans le système.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour accéder à cette fonctionnalité.
- Postconditions : Les chemins existants entre les deux noeuds spécifiés sont affichés à l'utilisateur.

Afficher les chemins possibles entre deux noeuds vérifiant les critères de l'EAR-RP

- Description : Ce cas d'utilisation permet à un utilisateur d'afficher les chemins possibles entre deux noeuds qui vérifient les critères de l'EAR-RP (Energy-Aware Routing with Residual Power).
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour accéder à cette fonctionnalité.
- Postconditions : Les chemins possibles entre les deux noeuds spécifiés, qui respectent les critères de l'EAR-RP, sont affichés à l'utilisateur

Afficher le chemin optimal selon l'EAR-RP

- Description : Ce cas d'utilisation permet à un utilisateur d'afficher le chemin optimal entre deux noeuds selon l'EAR-RP (Energy-Aware Routing with Residual Power).
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour accéder à cette fonctionnalité.
- Postconditions : Le chemin optimal entre les deux noeuds spécifiés, en suivant les critères de l'EAR-RP, est affiché à l'utilisateur.

Afficher le chemin optimal selon Dijkstra

- Description : Ce cas d'utilisation permet à un utilisateur d'afficher le chemin optimal entre deux noeuds selon l'algorithme de Dijkstra.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour accéder à cette fonctionnalité.
- Postconditions : Le chemin optimal entre les deux noeuds spécifiés, calculé en utilisant l'algorithme de Dijkstra, est affiché à l'utilisateur.

Router un paquet entre deux nœuds selon l'algorithme de Dijkstra

- Description : Ce cas d'utilisation permet à un utilisateur de router un paquet de données entre deux nœuds en utilisant l'algorithme de Dijkstra pour déterminer le chemin optimal.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour router un paquet entre les nœuds spécifiés.
- Postconditions : Le paquet de données est routé avec succès entre les deux nœuds en utilisant le chemin optimal déterminé par l'algorithme de Dijkstra.

Simuler l'activité du réseau avec l'EAR-RP

- Description : Ce cas d'utilisation permet à un utilisateur de simuler l'activité du réseau en utilisant l'algorithme EAR-RP (Energy-Aware Routing with Residual Power) pour évaluer la consommation d'énergie des nœuds du réseau.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour effectuer la simulation.
- Postconditions : La simulation de l'activité du réseau avec l'EAR-RP est exécutée et les résultats de la consommation d'énergie des nœuds sont affichés à l'utilisateur.

Simuler l'activité du réseau avec l'algorithme de Dijkstra

- Description : Ce cas d'utilisation permet à un utilisateur de simuler l'activité du réseau en utilisant l'algorithme de Dijkstra pour évaluer les performances de routage.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour effectuer la simulation.
- Postconditions : La simulation de l'activité du réseau avec l'algorithme de Dijkstra est exécutée et les résultats des performances de routage sont affichés à l'utilisateur.

Afficher les énergies résiduelles des différents nœuds du réseau

- Description : Ce cas d'utilisation permet à un utilisateur d'afficher les énergies résiduelles des différents nœuds du réseau, ce qui permet de surveiller l'état de l'énergie des nœuds.
- Acteurs : Utilisateur
- Préconditions : L'utilisateur est authentifié et a les autorisations nécessaires pour accéder à cette fonctionnalité.
- Postconditions : Les énergies résiduelles des différents nœuds du réseau sont affichées à l'utilisateur, permettant ainsi de vérifier leur état d'énergie.

3.4 Diagramme de séquence technique

Ajouter un point d'accès

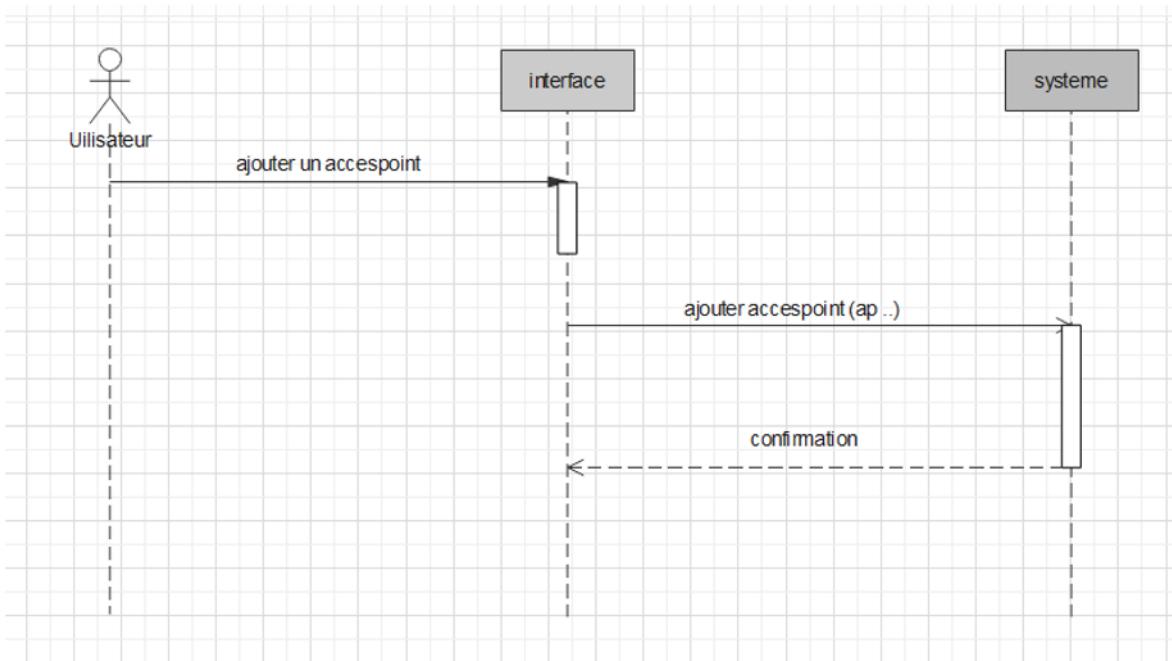


FIGURE 3.3 – Ajouter un point d'accès

Ajouter un lien

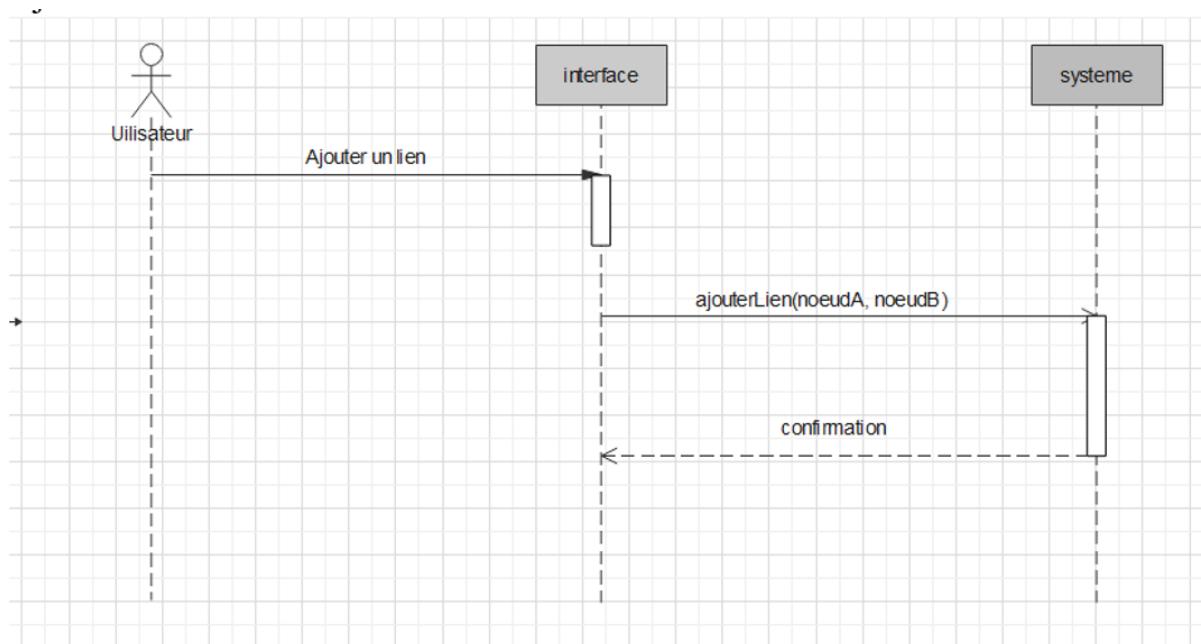


FIGURE 3.4 – Ajouter un lien

Créer une station

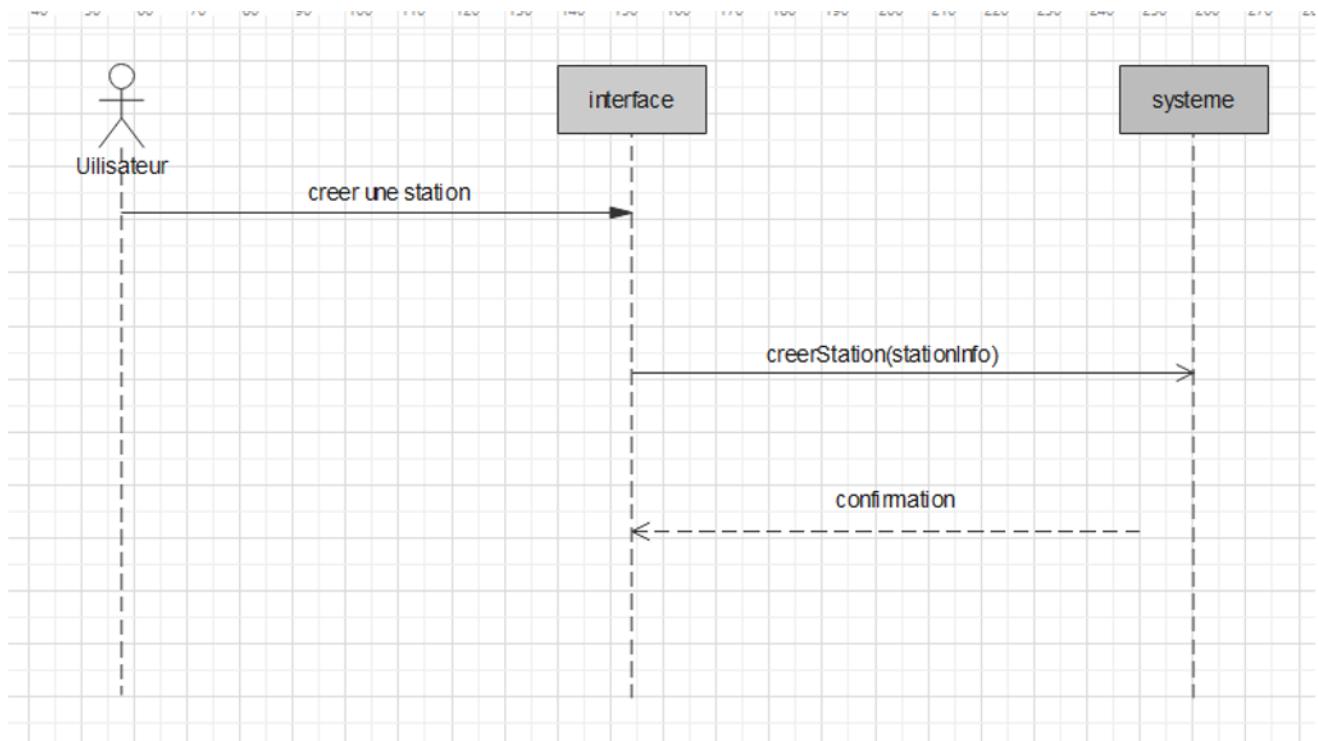


FIGURE 3.5 – Créer une station

Supprimer une station

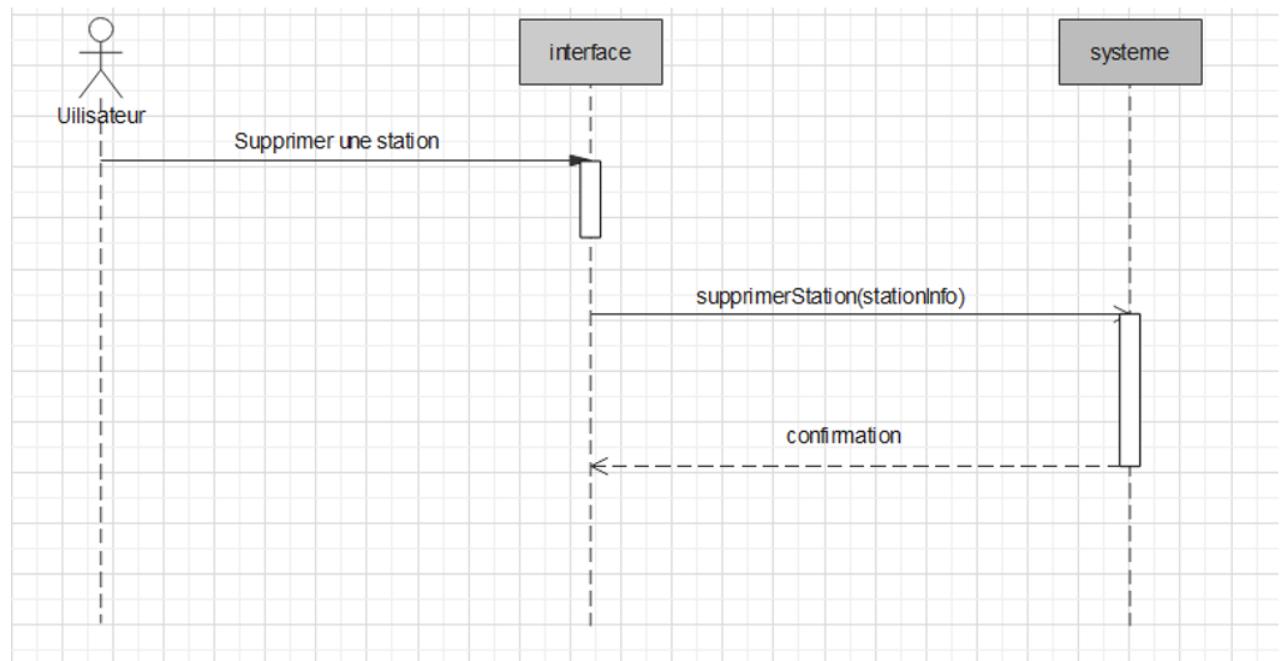


FIGURE 3.6 – Supprimer une station

3.5 Diagramme d'activités

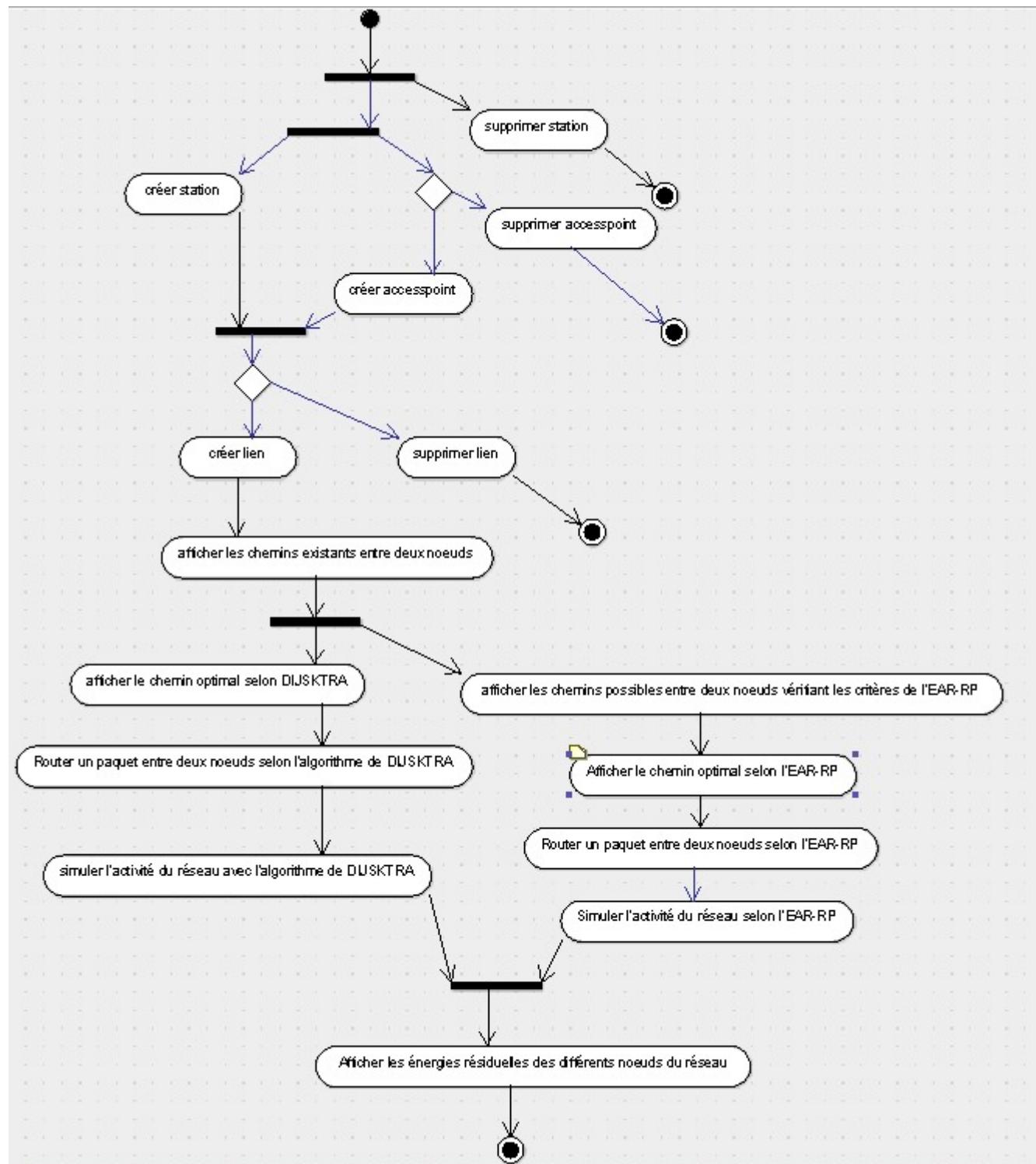


FIGURE 3.7 – Diagramme d'activités

Rapport sur la modélisation

1 Programmation linéaire en nombres entiers

La programmation linéaire en nombres entiers est une technique d'optimisation mathématique qui peut être utilisée pour résoudre une grande variété de problèmes. Les problèmes de PLNB sont généralement modélisés comme un système d'équations linéaires et d'inégalités, dans le but de trouver une solution qui satisfait toutes les contraintes et optimise une fonction objectif donnée. les problèmes de PLNB peuvent être difficiles à résoudre, et un certain nombre de défis peuvent survenir lorsque l'on tente de résoudre un problème L'un des défis est que les problèmes peuvent être très vastes et complexes : Cela peut rendre difficile la recherche d'une solution au problème, et cela peut également rendre difficile la compréhension de la solution trouvée. Un autre défi est que les problèmes peuvent être NP-difficiles : Cela signifie qu'il n'existe aucun algorithme efficace connu pour résoudre le problème et que le mieux que l'on puisse faire est de trouver une solution proche de la solution optimale. La programmation linéaire est une branche de l'optimisation mathématique qui vise à maximiser ou minimiser une fonction linéaire sous des contraintes linéaires. Dans le cas de la programmation linéaire en nombres entiers, les variables de décision sont restreintes à prendre des valeurs entières. Voici les étapes générales pour résoudre un problème de PLNE :

- Formulation du problème : Tout d'abord, il est important de formuler clairement le problème sous forme mathématique. Cela implique de définir les objectifs du problème, les variables de décision, les contraintes et la fonction objective.
- Modélisation mathématique :En utilisant les informations fournies dans la formulation du problème, vous devez traduire ces informations en équations linéaires et inégalités. Chaque variable de décision doit être définie, ainsi que les contraintes qui lui sont associées.
- Fonction objective :Déterminez si vous souhaitez maximiser ou minimiser une fonction linéaire spécifique. Cette fonction représente l'objectif du problème et dépend des variables de décision.
- Contraintes :Identifiez toutes les contraintes du problème, telles que des limites sur les ressources disponibles ou des relations entre les variables. Chaque contrainte doit être exprimée sous forme d'équation ou d'inégalité linéaire.
- Linéarisation :Assurez-vous que toutes les équations et inégalités sont linéaires. Si des termes non linéaires sont présents, vous devrez les linéariser en introduisant de nouvelles variables ou en utilisant des techniques spécifiques, telles que la linéarisation de McCormick.
- Résolution du problème :Utilisez un solveur PLNE pour résoudre le problème. Il existe de nombreux solveurs disponibles, tels que CPLEX, Gurobi ou SCIP, qui sont capables de résoudre des problèmes de PLNE de grande taille et de manière efficace.

- Interprétation des résultats : Analysez la solution obtenue pour prendre des décisions éclairées. Vérifiez si les contraintes sont satisfaites, examinez la valeur de la fonction objective et interprétez les valeurs des variables de décision.
- Vérification de la faisabilité : Dans certains cas, il se peut que la solution trouvée ne soit pas réalisable en pratique. Vous devrez peut-être apporter des ajustements à votre modèle ou revoir les contraintes pour obtenir une solution réalisable.

2 Contexte

Les zones rurales et difficiles d'accès se caractérisent par un manque ou une alimentation électrique irrégulière qui constraint les noeuds du réseau à fonctionner sur batteries. Dans ce texte, les noeuds du réseau ont une capacité énergétique limitée. Compte tenu de cette contrainte et d'éléments tels que le coût économique et la protection de l'environnement, il est important d'améliorer la consommation d'énergie dans les réseaux.

SDN est une architecture réseau émergente qui sépare le plan de contrôle du plan de données. Le plan de contrôle est placé dans un contrôleur centralisé qui a une vue globale de la topologie du réseau. Le plan de données est composé de routeurs et de commutateurs qui relaient les données. Des protocoles tels que le protocole OpenFlow [5] sont utilisés pour traduire les politiques de haut niveau en règles qui seront placées dans les tables de flux des routeurs. Ils utilisent ces tables pour relayer les messages dans le réseau. Plusieurs services tels que la surveillance du réseau, la facturation, le routage, le contrôle d'accès et l'équilibrage de charge nécessitent des modèles de placement de politiques dédiés qui peuvent être facilement configurés. L'intégration des concepts SDN dans des réseaux où les noeuds ont une capacité d'énergie limitée peut nous

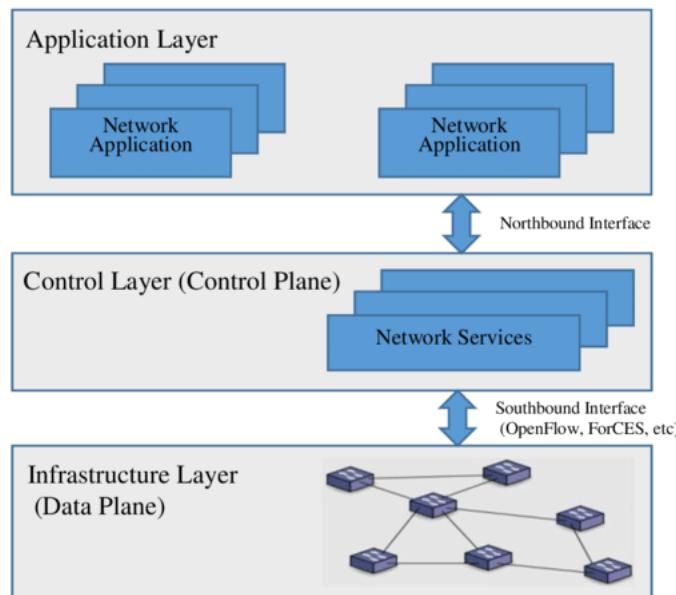


FIGURE 4.1 – Architecture générale des SDN

permettre d'optimiser la capacité énergétique qui est une contrainte importante à prendre en compte lors de la conception ou de la modélisation d'une solution de routage économique en énergie basée sur le placement de règles, et en particulier lors de l'attribution d'un chemin à chaque flux de données entrant. Dans un

réseau déployé dans une zone rurale où les routeurs sont autonomes en énergie, il est nécessaire de suivre la consommation d'énergie des noeuds en maintenant la batterie en vie le plus longtemps possible. L'objectif est d'augmenter la durée de vie des noeuds et du réseau en évitant les noeuds à faible énergie. Par conséquent, une solution efficace de placement de règles pour économiser l'énergie doit prendre en compte la consommation d'énergie tant du point de vue du réseau des noeuds. Du point de vue des nœuds, la meilleure route est celle qui évite les nœuds ayant une faible capacité énergétique.

Notre approche consiste à définir des règles de routage spécifiques dans le plan de contrôle en fonction des caractéristiques du réseau, puis à les placer sur les routeurs/commutateurs dans le plan de données. L'objectif ultime est de minimiser la consommation d'énergie totale en tenant compte des contraintes liées à l'énergie résiduelle des nœuds. Nous formalisons le problème sous la forme d'un programme linéaire entier (ILP), qui est une méthode d'optimisation robuste éprouvée dans le domaine [4, 2]. Nous définissons des algorithmes qui trouvent le chemin optimal à travers le réseau. Ces algorithmes sont des solutions heuristiques gloutonnes du programme linéaire entier avec une fonction objective qui minimise la consommation d'énergie totale sous contraintes. Le contrôleur doit résoudre le programme linéaire entier à chaque nouveau flux de données arrivant dans le réseau. Cette approche n'est pas réalisable en pratique car elle peut créer une surcharge fréquente dans le réseau en raison des mises à jour fréquentes de la table de flux [4]. Notre approche améliore ces coûts de calcul supplémentaires en calculant les chemins uniquement lorsque c'est nécessaire, c'est-à-dire lorsque aucun chemin n'existe ou s'il existe, il ne satisfait pas les contraintes.

2.1 Architecture étudiée dans notre modélisation : réseau maillé sans fils

Une architecture de réseau maillé sans fil (Wireless Mesh Network - WMN) est un type de réseau sans fil dans lequel plusieurs nœuds interconnectés forment une infrastructure de communication. Contrairement aux réseaux traditionnels où les appareils se connectent directement à un point d'accès central, dans un réseau maillé sans fil, chaque nœud peut servir de point d'accès et de relais pour d'autres nœuds.

Dans un réseau maillé sans fil, les nœuds sont généralement équipés de capacités de routage, ce qui leur permet de transmettre les données à travers le réseau en choisissant les itinéraires les plus efficaces. Cette approche de routage multi-sauts permet d'étendre la portée du réseau en reliant les nœuds les uns aux autres, formant ainsi une infrastructure auto-organisée et résiliente.

Les réseaux maillés sans fil sont largement utilisés dans diverses applications, notamment dans les zones rurales ou les zones urbaines denses où le déploiement de l'infrastructure câblée est coûteux ou difficile. Ils offrent une connectivité étendue, une tolérance aux pannes et une flexibilité pour ajouter ou déplacer des nœuds sans perturber l'ensemble du réseau.

Les réseaux maillés sans fil ont des applications dans les domaines tels que l'accès à Internet, la vidéo-surveillance, les réseaux de capteurs, les réseaux de ville intelligente, et bien d'autres. Ils permettent une connectivité sans fil étendue, fiable et adaptable, offrant ainsi des avantages significatifs dans la mise en réseau et la communication dans divers environnements.

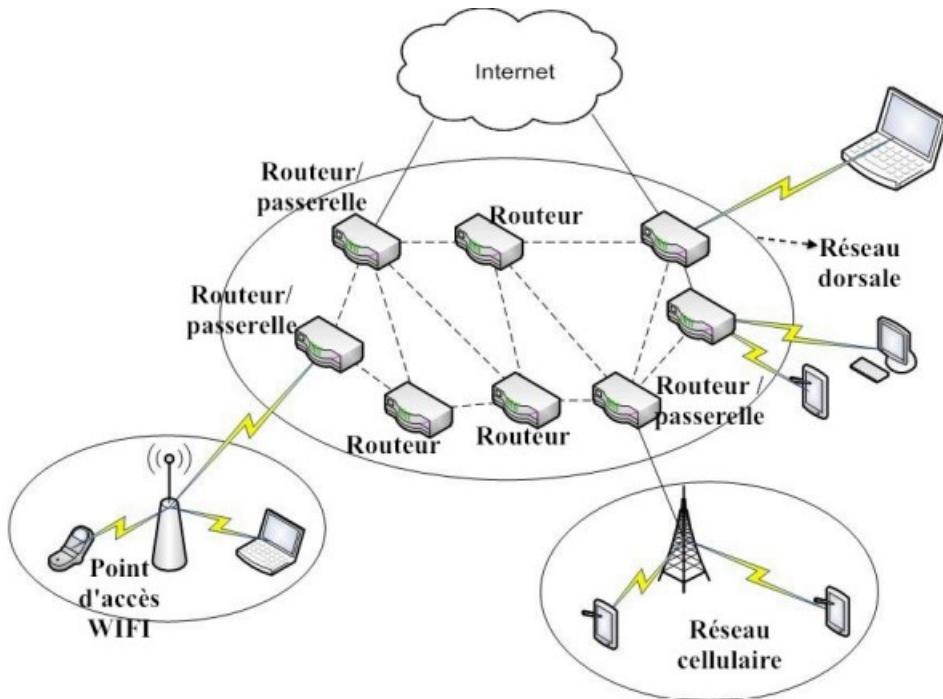


FIGURE 4.2 – Architecture du réseau maillé : les points d'accès collectent le trafic des clients mobiles/fixes et l'envoient vers les passerelles connectées aux réseaux externes qui fournissent divers services

3 Modélisation EAR-RP

3.1 Vue d'ensemble d'EAR-RP

Le problème du routage économique en énergie dans les réseaux sans fil programmables consiste à trouver un chemin qui maximise l'énergie résiduelle des nœuds intermédiaires et minimise la consommation totale d'énergie lors de la transmission de l'énergie de la source à la destination. La politique de routage d'un nœud source vers un nœud destination est définie de manière réactive et placée dans la table de flux du routeur par un contrôleur pour chaque nouveau flux. L'inconvénient peut être une surcharge du réseau, une latence élevée et une consommation d'énergie élevée.

En revanche, dans le cadre de l'EAR-RP (Energy-Aware Routing with Reactive Policy), le contrôleur calcule un nouveau chemin uniquement s'il est nécessaire. Lorsqu'un nouveau flux de données arrive, le nœud source vérifie s'il existe une entrée disponible dans la table de flux pour relayer ce flux et en informe le contrôleur. Si une entrée existe, le contrôleur vérifie si les nœuds participant au processus de routage disposent d'une énergie résiduelle et d'une capacité mémoire suffisantes pour relayer le nouveau flux de données. Si c'est le cas, le contrôleur demande au nœud source d'utiliser cette entrée, sinon il calcule un nouveau chemin pour cette entrée. Si aucune entrée ne correspond au nouveau flux de données, le contrôleur calcule un nouveau chemin et met à jour les tables de flux des nœuds qui participeront au routage.

3.2 Modèle de conception EAR-RP

Le réseau est représenté comme un graphe non orienté $G = (V, E)$ où V représente l'ensemble des nœuds et E l'ensemble des liens sans fil. Nous distinguons quatre types de nœuds dans le réseau :

1. les stations mobiles (STAs) qui représentent les utilisateurs finaux qui consomment le service. Ils ne

sont pas inclus dans le réseau maillé.

2. Les points d'accès maillés (MAPs) qui fournissent simultanément les fonctionnalités de point d'accès maillé et de point d'accès.
3. Le contrôleur, qui permet de centraliser la gestion du réseau.

Consommation d'énergie Un nœud $v_k \in V$ dispose d'au moins trois interfaces de communication :

1. L'interface du point d'accès pour la communication avec les clients ;
2. L'interface maillée pour relayer la communication au sein du réseau maillé,
3. et l'interface de contrôle pour la communication avec le contrôleur.

À un moment donné, chaque noeud v_k a une énergie résiduelle qui correspond au niveau de la batterie à ce moment précis noté Er_k .

Traffic Pour un ensemble de flux de données F , une solution de placement de règles consiste à assigner à chaque flux f_l un chemin P_l tel que les contraintes de capacité énergétique et de taille de mémoire soient respectées. Les routeurs SDN utilisent des tables de flux implémentées dans la mémoire TCAM pour prendre des décisions de routage. Une entrée est définie par une règle de correspondance et une action à effectuer. Lorsqu'il reçoit un paquet, le dispositif de routage sélectionne la règle correspondante avec la plus haute priorité et exécute l'action correspondante. Un paquet sans action correspondante est traité en fonction de la règle par défaut ou envoyé au contrôleur.

4 Programme linéaire en nombres entiers du EAR-RP

L'objectif de l'EAR-RP (Energy-Aware Routing with Reactive Policy) est de minimiser la consommation totale d'énergie entre un nœud source s et un nœud destination d en tenant compte de l'énergie résiduelle et de la capacité mémoire des nœuds pour chaque nouveau flux de données. Le tableau 1 ci-dessous récapitule les différentes variables utilisées. Nous avons formalisé le problème sous la forme d'un Programme Linéaire

Notation	Signification
$E_{T(s,d,l)}$	Energie totale consommée de la source s à la destination d pour le flux de données f_l
Er_{v_k}	Énergie résiduelle du nœud v_k
x_{jkl}	Booléen qui indique si le nœud v_j est choisi pour le flux de données f_l
$E_{(v_j,v_k,l)}$	Coût d'énergie à partir du nœud v_j jusqu'au nœud v_k pour le flux de données f_l
$V^+(v_j)$	Ensemble des nœuds voisins au nœud v_j
s	Nœud source
d	Nœud destinataire

TABLE 4.1 – Notations utilisées

Entier (PLE). La fonction objectif vise à minimiser la consommation totale d'énergie d'un nœud source s à un nœud destination d pour un flux f_l , sous certaines contraintes.

Fonction objective

$$\text{Min}E_{T(s,d,l)} = \sum_{v_j \in V, v_k \in V^+(v_j), f_l \in F} E_{(v_j, v_k, l)} x_{jkl}$$

Soumise aux contraintes :

$$x_{jkl} \in \{0, 1\}$$

$$x_{jlk} = \begin{cases} 1 & \text{Si le nœud } v_j \text{ choisit le nœud } v_k \text{ pour relayer les informations du flot de données } f_l \\ 0 & \text{sinon} \end{cases} \quad (4.1)$$

$$\forall v_j \in V, \forall v_k \in V^+(v_j), \forall f_l \in F, \sum x_{jkl} \leq 1 \text{ si } v_j \neq D \quad (4.2)$$

$$\forall v_j \in V, \forall v_k \in V^+(v_j), \forall f_l \in F, \sum x_{jkl} = -1 \text{ si } v_j = D \quad (4.3)$$

$$\forall v_j \in V, v_k \in V^+(v_j), Er_{v_k} > Er_{threshold} \quad (4.4)$$

$$\forall v_j \in V, v_k \in V^+(v_j), Cm_{v_k, t} < Cm_{max}, t \in \mathbb{N} \quad (4.5)$$

Pour résoudre la fonction objectif définie dans l'équation (1), nous proposons une heuristique gloutonne appelée EAR-RP (routage économique en énergie basé sur le placement de règles) qui prend en entrée un réseau sans fil programmable, un nœud source s, un nœud destination d, un flux de données f_l et renvoie un itinéraire optimal satisfaisant les contraintes. EAR-RP est un algorithme heuristique en trois étapes :

Étape 1 : L'heuristique identifie un sous-ensemble d'itinéraires qui respectent les contraintes d'énergie résiduelle. Ici, EAR-RP utilise l'algorithme de recherche en profondeur (DFS) pour trouver tous les itinéraires possibles de la source à la destination. Ensuite, l'algorithme évalue chaque itinéraire et ne conserve que l'ensemble de nœuds satisfaisant les contraintes.

Étape 2 : L'algorithme calcule l'énergie totale de chaque itinéraire sélectionné à l'étape 1 et renvoie l'itinéraire ayant une énergie totale minimale. Il consiste à calculer la somme de l'énergie d'envoi et de réception nécessaire à chaque nœud dans chaque itinéraire, puis à sauvegarder l'itinéraire ayant une énergie totale minimale. Nous supposons que la valeur d'énergie de transmission et de réception est constante pour chaque nœud. Il en va de même lors de la minimisation du nombre de liens. Cet itinéraire est l'itinéraire optimal.

Étape 3 : EAR-RP installe les nouvelles règles dans la table de flux de chaque nœud appartenant à l'itinéraire optimal, et les nœuds les utilisent pour relayer les informations de la source à la destination.

5 EAR-RP : évaluation et discussion

5.1 Architecture logique

SDN permet aux administrateurs réseau de développer un protocole de routage souhaitable sur le contrôleur pour rendre le réseau plus efficace et plus gérable.

print energies : Ce module affiche l'énergie résiduelle des nœuds après une réception ou une transmission. Nous avons fixé la valeur de consommation d'énergie de transmission et de réception à 0,5 Wh et 0,5 Wh respectivement. Les valeurs initiales des énergies résiduelles étant fixées à 5 Wh

Algorithm 1 EAR-RP Algorithm

1: **Procédure find_all_possible_paths****Require:** Graph SD-WCN, Source s, Destination d, Flux F**Sortie** un ensemble de routes possibles P^* **Début** $P^* = \{\}$ $P \leftarrow$ Toutes les routes possibles de s à d //utiliser l'algorithme de recherche Depth-firstPour tout $P_i \in P$ faire si ($\forall v_k \in P_i, Er_{v_k} > Er_{threshold}$ et $Cm_{v_k,t} < Cm_{max}$) $P^* \leftarrow P_i$ retourne P^*

Fin pour

Fin2: **Procédure find_optimal_path****Require:** Un ensemble P^* des routes possibles qui satisfont les contraintes**Sortie** Une route optimal $P_{optimal}$ qui a une énergie totale minimale**Début** $Er_{min} \leftarrow ET(P^*[0])$ $P_{optimal} \leftarrow P^*[0]$ Pour tout $P_i \in P^*$ faire Calculer $ET(P_i)$ si ($ET(P_i) < Er_{min}$) $P_{optimal} \leftarrow P_i$ $Er_{min} \leftarrow ET(P_i)$

Fin pour

retourne $P_{optimal}$ **Fin**3: **Procédure Update_Flow_Route****Require:** Optimal route $P_{optimal}$ **Sortie** $\forall v_k \in P_{optimal}$, Table des flots mise à jour**Début**Pour tout $v_k \in P_{optimal}$ faire

Installer une nouvelle entrée dans la table des flots

Fin pour

Fin

Algorithm 2 Recherche en profondeur

1: **Procédure find_all_paths****Require:** Graph SD-WCN, Source s, Destination d, Flux F**Début** $R = \{\}$

chemin={ }

Depth_first(graphe G, sommet t)

listNoeudsMarques={ }

listNoeudsMarques.ajouter(t)

Ajouter s dans chemin

Si t == d alors

chemin.ajouter('l')

chemin.ajouter(s)

retourne

Pour tout sommet v voisin de s

Si v n'est pas dans listNoeudsMarques alors

Depth_first (G,v)

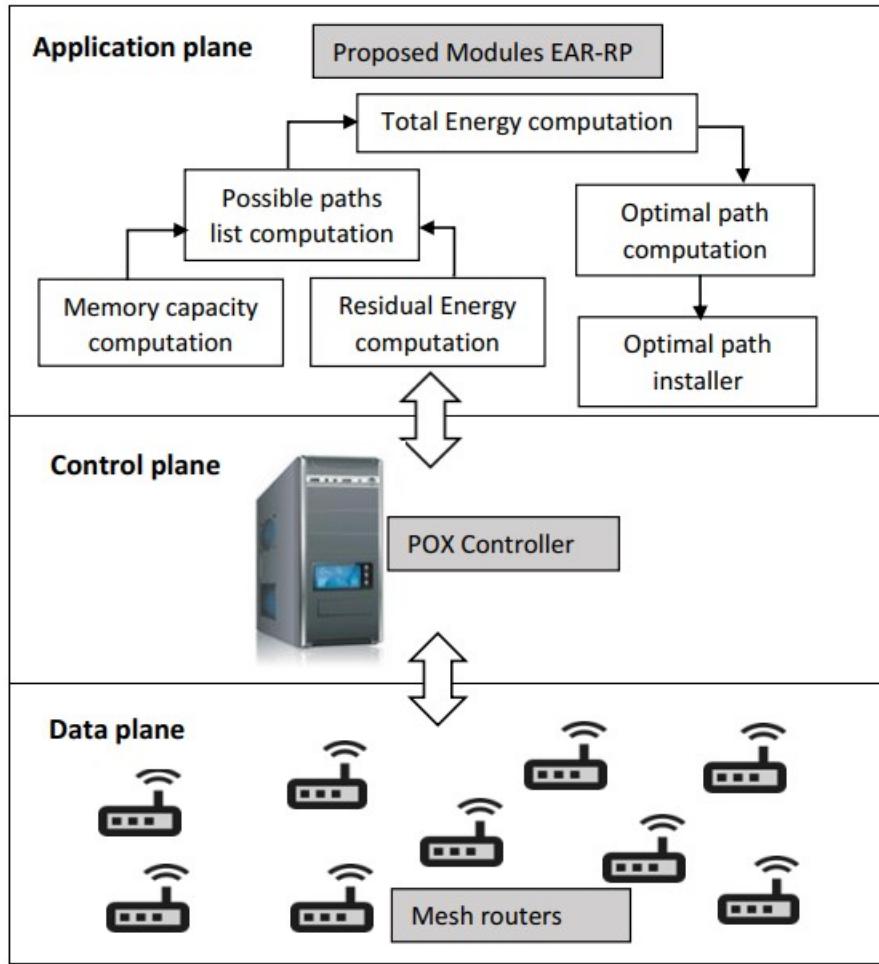


FIGURE 4.3 – logical architecture of EAR-RP

Possible paths list computation : Ce module liste les chemins possibles en évitant les nœuds ayant une énergie résiduelle inférieure à la valeur seuil, conformément aux équations 4.4 du programme linéaire en nombres entiers.

Total energy computation : Ce module reçoit un chemin de la matrice PossiblePaths et calcule la consommation d'énergie totale, puis renvoie le résultat.

Optimal path computation : Ce module reçoit la matrice PossiblePaths et appelle le calcul de l'énergie totale pour calculer la consommation d'énergie totale de chaque chemin, puis renvoie le vecteur OptimalPath, qui correspond au chemin ayant la valeur minimale.

Optimal path installer : Ce module installe les règles de routage OpenFlow dans chaque nœud du vecteur OptimalPath. Le contrôleur exécute les modules mentionnés ci-dessus à chaque fois qu'un nouveau flux entre dans le réseau afin de trouver le chemin optimal en fonction de l'état actuel du réseau, en tenant compte de l'énergie résiduelle de chaque nœud, ainsi que de l'énergie totale du chemin.

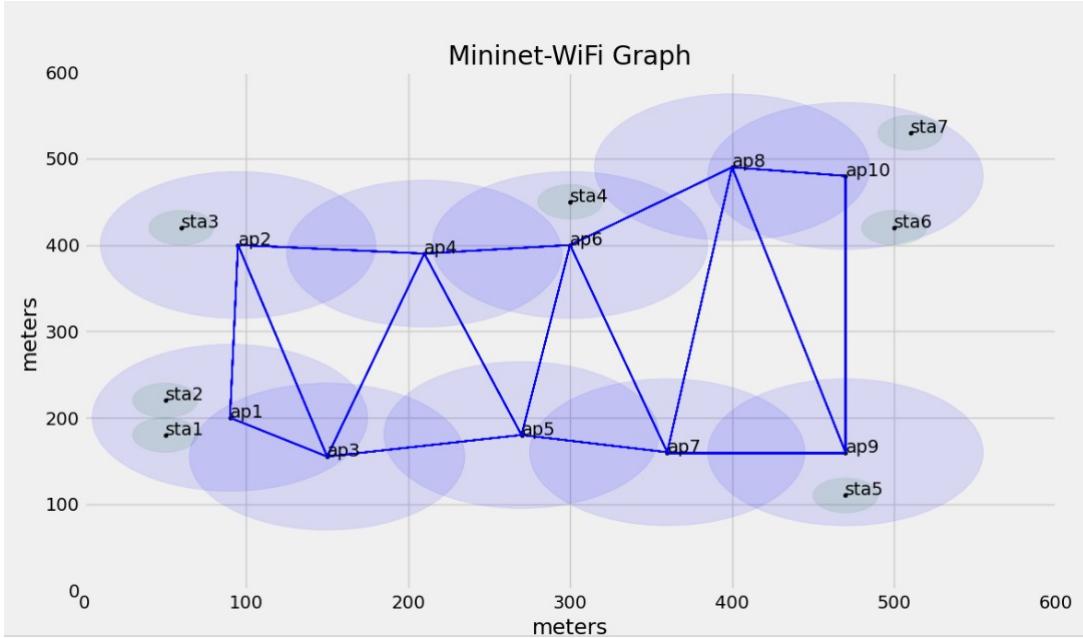


FIGURE 4.4 – Topologie du réseau dans le plan de données (Mininet-Wifi)

5.2 Mesures de simulation et d'évaluation des performances

Dans cette section, nous réalisons une série de simulations pour évaluer EAR-RP et comparer ses performances avec l'algorithme du plus court chemin de Dijkstra. L'algorithme de Dijkstra calcule le plus court chemin en fonction de l'état des liens. Pour l'évaluation expérimentale, nous avons utilisé un émulateur SDN Mininet-wifi [25]. Il permet la création d'un réseau virtuel composé de points d'accès SDN, de stations (hôtes) interconnectés par des canaux sans fil (liens). Mininet-wifi s'exécute dans VirtualBox sur le système d'exploitation Linux Ubuntu 18.04 LTS. Le langage de programmation Python est utilisé pour mettre en œuvre l'architecture proposée. Le tableau 4.2 résume les outils utilisés pour construire le prototype. La

Logiciel	Purpose	Version
Ubuntu Linux	Hosting OS	18.04
Mininet-wifi	SD-WCN Emulator	2.3.7dev
Python	Programming Language	2.7

TABLE 4.2 – Logiciels utilisés dans l'expérimentation

performance d'EARP-RP est évaluée à l'aide des mesures suivantes :

- **Durée de vie** : Il s'agit du laps de temps entre le début de l'opération du réseau et le moment où le premier noeud épouse son énergie.
- **Consommation totale d'énergie** : Cette mesure représente le pourcentage d'énergie totale consommée depuis une source jusqu'à une destination pour un flux de données.

5.3 Résultats et discussion

Dans cette section, nous analysons et discutons les résultats obtenus à partir de différentes simulations pour les algorithmes EAR-RP et Dijkstra dans différents scénarios. La figure 4.5 montre que l'EAR RP surpassé Dijkstra en termes de durée de vie. L'EAR-RP utilise la valeur d'énergie résiduelle de chaque routeur

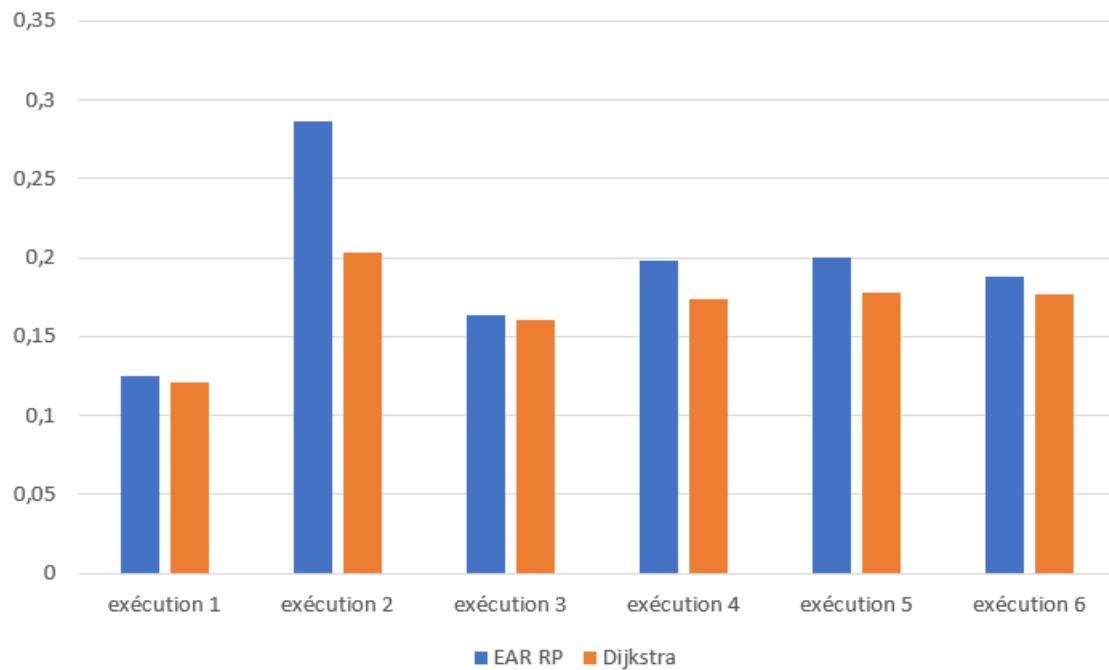


FIGURE 4.5 – Durée de vie lors des différentes simulations

Exécution	Trajets des paquets envoyés
1	sta3-sta7, sta4-sta6, sta5-sta4, sta6-sta7, sta2-sta6
2	sta2-sta7, sta3-sta2, sta1-sta6, sta1-sta2, sta2-sta6, sta1-sta4
3	sta4-sta3, sta6-sta2, sta4-sta5, sta3-sta1, sta4-sta7, sta2-sta3
4	sta2-sta5, sta3-sta2, sta4-sta7, sta7-sta2, sta5-sta6, sta5-sta3
5	sta3-sta7, sta7-sta5, sta5-sta1, sta6-sta3, sta3-sta5
6	sta6-sta7, sta1-sta5, sta5-sta6, sta5-sta1, sta6-sta2, sta3-sta4

TABLE 4.3 – Détails sur la figure 4.5

dans le calcul des chemins possibles selon la contrainte (4.4) dans le PLNE. Les routeurs dont l'énergie résiduelle est inférieure à la valeur seuil ne peuvent pas être utilisés tant qu'il existe des solutions alternatives possibles. Cette technique permet d'économiser de l'énergie dans les routeurs et de prolonger la durée de vie du réseau. Pour évaluer la consommation totale d'énergie d'une source à une destination, nous avons utilisé la topologie de réseau dans la figure 5. Elle est composée de dix points d'accès maillés (ap1 - ap10) et de sept stations mobiles (sta1 - sta7). Nous considérons que la station mobile est fixe pendant le test. Les scénarios ont été réalisés avec des flux de données ping : 2 paquets, 4 paquets et 5 paquets avec les algorithmes EARRP et Dijkstra. Les figures 4.6, 4.7 et 4.8 montrent les résultats obtenus. Nous remarquons que pour un flux de 2, 4 paquets, le pourcentage d'énergie consommée est le même pour les deux algorithmes car tous les nœuds ont une valeur d'énergie résiduelle supérieure à la valeur seuil. Dans ce cas, l'algorithme EAR-RP choisit le chemin qui a la plus petite valeur d'énergie totale, ce qui correspond évidemment au chemin le plus court en termes d'énergie. À partir de 5 paquets, l'algorithme EAR-RP consomme plus d'énergie car certains nœuds ont une valeur d'énergie inférieure à la valeur seuil et, pour les maintenir en vie, l'algorithme EAR-RP choisit un chemin plus long. Cependant, Dijkstra maintient le chemin le plus court, ce qui a pour

conséquence que ces nœuds s'éteindront après un certain temps. Dans un réseau à contrainte énergétique, l'objectif est de maintenir les nœuds en vie le plus longtemps possible.

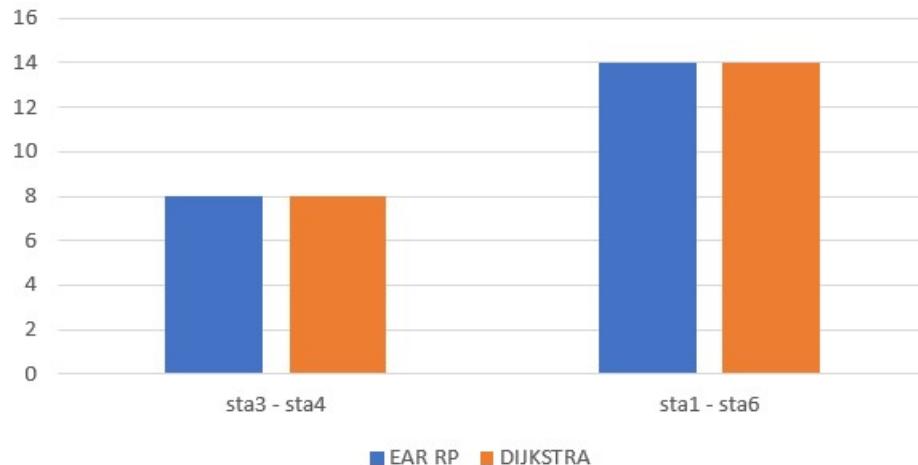


FIGURE 4.6 – Energie totale consommée pour 2 paquets

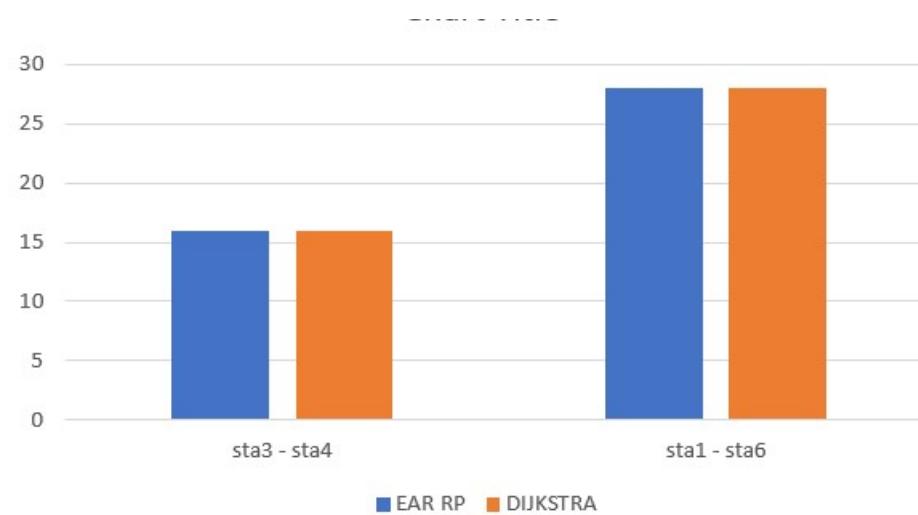


FIGURE 4.7 – Energie totale consommée pour 4 paquets

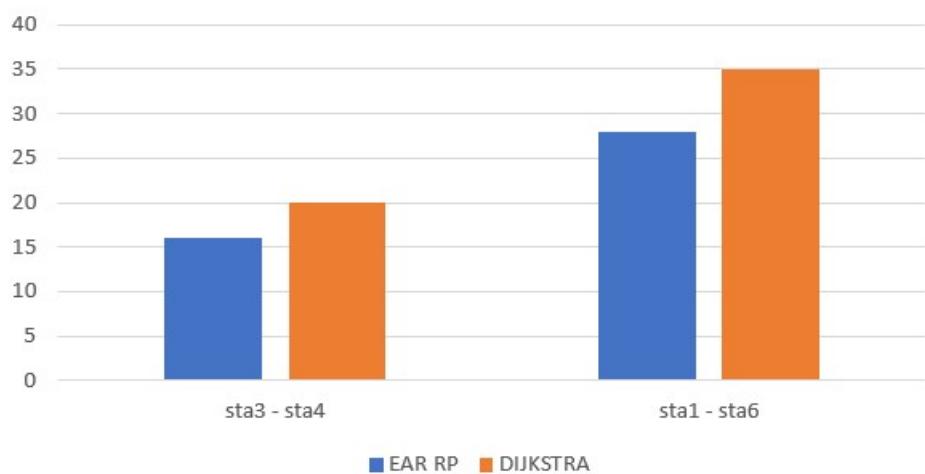


FIGURE 4.8 – Energie totale consommée pour 5 paquets

Rapport sur l'implémentation

Dans cette partie nous allons définir le simulateur utilisé «MININET-WIFI » ensuite nous allons présenter notre implémentation qui consiste à trouver le chemin optimal en tenant compte uniquement de la quantité d'énergie consommé c'est-à-dire trouver le chemin qui minimise l'énergie totale consommée de la source a la destination et évaluer la durée de vie des nœuds après un temps t d'échange des messages

1 Mininet-Wifi

Mininet-WiFi : est un fork de l'émulateur de réseau Mininet SDN . Les développeurs de Mininet-wifi ont étendu les fonctionnalités de Mininet en ajoutant des stations wifi virtualisées et des points d'accès basés sur les pilotes sans fil Linux standard. Ils ont également ajouté des classes pour prendre en charge l'ajout de ces dispositifs sans fil dans un scénario de réseau Mininet et pour émuler les attributs d'une station mobile tels que la position et le mouvement par rapport aux points d'accès.

Le Mininet-WiFi a étendu le code Mininet de base en ajoutant ou en modifiant des classes et des scripts. Ainsi, Mininet-wifi ajoute de nouvelles fonctionnalités et prend toujours en charge toutes les capacités d'émulation SDN normale de l'émulateur de réseau Mininet standard.

2 Etapes de la simulation

2.1 Création de la topologie

La topologie utilisée dans notre projet se compose de 10 point d'accès, un contrôleur et 7 stations .Cette topologie a été créée grâce au simulateur Mininet-wifi. Après avoir lancé ce dernier, nous avons créé une topologie sous forme de fichier python Il comporte les parties suivantes :

1. **Importation de quelques bibliothèques de mininet-wifi**
2. **Création de contrôleurs** : on crée un contrôleur avec le nom 'c0' et l'adresse IP '0.0.0.0'. Le contrôleur est de type RemoteController et utilise le protocole TCP avec le port 6634.
3. **Configuration des noeuds du réseau**

```
from mininet.log import setLogLevel, info
from mn_wifi.net import Mininet_wifi
from mn_wifi.node import OVSKernelAP, Station
from mn_wifi.link import wmediumd
from mn_wifi.wmediumdConnector import interference
from mininet.cli import CLI
from mininet.node import Controller, RemoteController
```

FIGURE 5.1 – Bibliothèques nécessaires pour la création de la topologie

```
c0 = net.addController('c0', ip='0.0.0.0', controller=RemoteController, protocol='tcp', port=6634)
```

FIGURE 5.2 – Bibliothèques nécessaires pour la création de la topologie

```
accessPointsPositions = ["90,200,0", "95,400,0", "150,155,0", "210,390,0", "270,180,0", "300,400,0", "360,160,0", "400,490,0", "470,160,0", "530,400,0"]
aps = []
for i in range(1, 11):
    ap = net.addAccessPoint('ap%d' % i, cls=OVSKernelAP, ssid='ap%d-ssid' % i, channel='1', mode='g', position=accessPointsPositions[i-1], f=aps.append(ap))
```

FIGURE 5.3 – Crédit des points d'accès

4. Ajout des liens entre les noeuds du réseau

5. Affichage du

- 6. Création des points d'accès (access points) :** dans notre topologie on a 10 points d'accès , au qu'elle nous avons attribué les adresses IP de 127.0.0.1 a 127.0.0.7 et les coordonnées permettant de placer les nœuds dans le graphes

2.2 Lancement de la topologie

La fonction nous permettant de créer cette topologie est le suivant : **myNetwork()** . Après avoir créé la topologie, on ouvre le terminal on entre dans le fichier nommer mininet-wifi puis examples après on lance notre fichier python " creation_topologie.py " grâce à la commande suivante :

sudo mn -c && sudo python creation_topologie Ensuite nous obtenons le mininet-wifi Graph .

Il est important de préciser que nous avons utilisé le module **net.plotGraph**

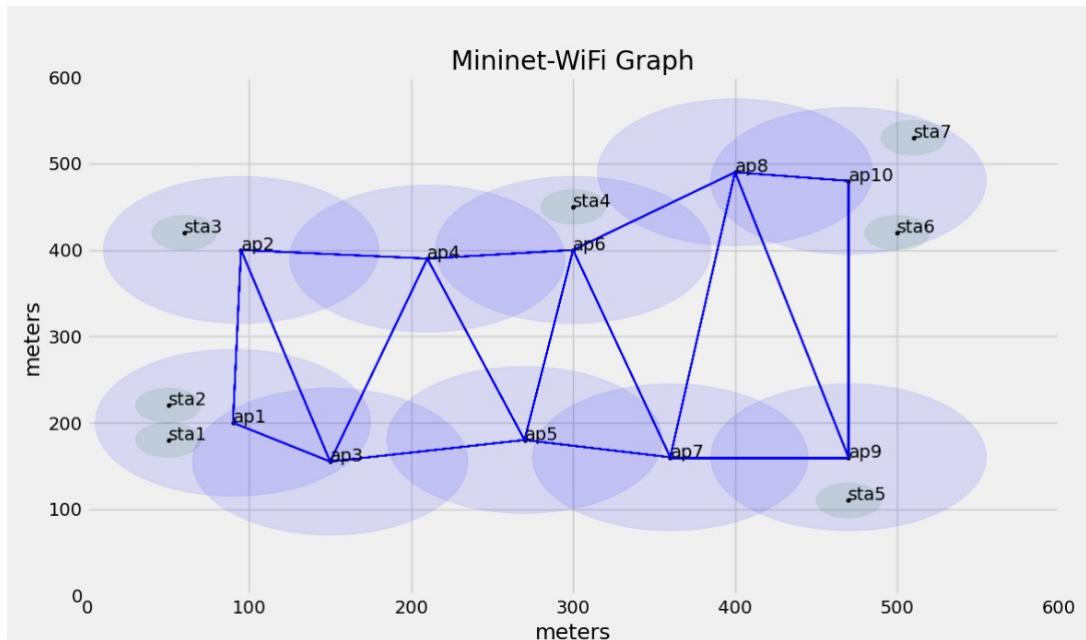


FIGURE 5.4 – Graphe(Mininet-wifi)

]

2.3 Recherche de tous les chemins existants entre deux nœuds

La fonction `find_all_paths(net, src, dst, aps, stations, path=[])` utilise une approche récursive pour trouver tous les chemins possibles entre les nœuds source (`src`) et destination (`dst`). Elle utilise la fonction `neighbors` pour obtenir les voisins du nœud source et continue à explorer les chemins jusqu'à atteindre le nœud de destination. La fonction renvoie une liste contenant tous les chemins trouvés.

```
def find_all_paths(net, src, dst, aps, stations, path=[]):
    path = path + [src]
    if src == dst:
        return [path]
    paths = []
    for neighbor in neighbors(src, aps, stations):
        if neighbor not in path:
            new_paths = find_all_paths(net, neighbor, dst, aps, stations, path)
            for new_path in new_paths:
                paths.append(new_path)
return paths
```

FIGURE 5.5 – Chercher tous les chemins existants

Nous pouvons apercevoir le résultat de cette fonction à travers notre interface graphique. Il suffit de remplir des zones de texte pour spécifier les noeuds source et destinataire et ensuite il faut cliquer sur le bouton "Afficher les chemins possibles". Ensuite nous pouvons voir les chemins se dessiner progressivement.

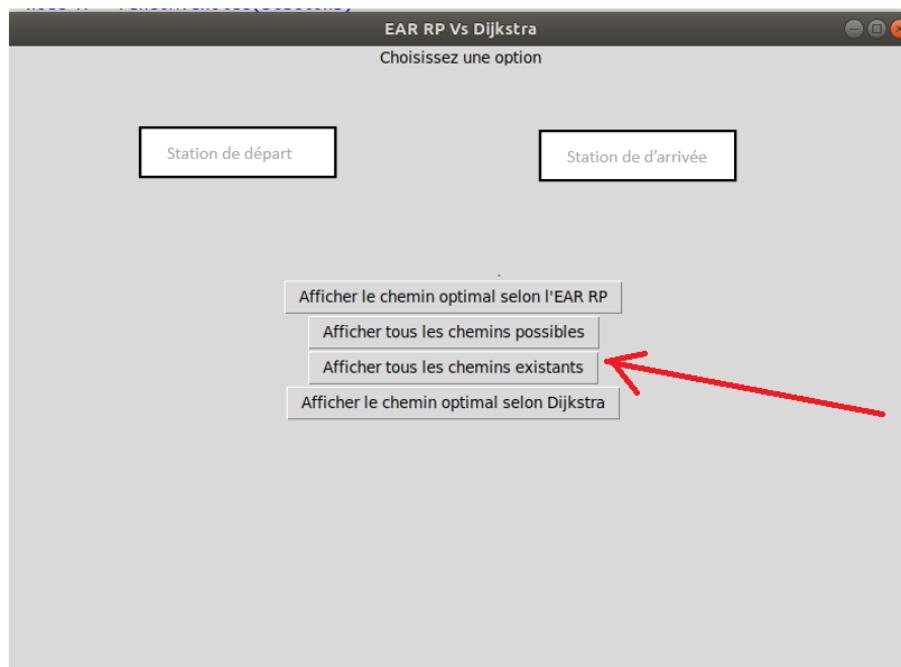


FIGURE 5.6 – Chercher tous les chemins existants

Voici le premier chemin qui s'est dessiné pour notre cas (`source=sta3, destination=sta5`). Nous pouvons voir le résultat à la figure 5.7) Le deuxième chemin peut alors s'afficher(nous avons choisi des couleurs différentes

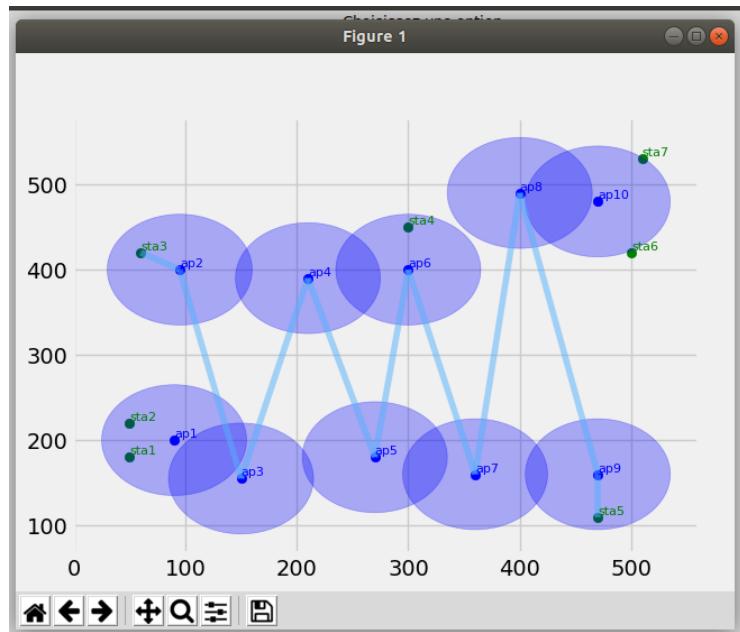


FIGURE 5.7 – Chercher tous les chemins existants(premier chemin)

pour chacun des chemins). On peut voir le résultat à la figure 5.8). A la fin, nous pouvons voir le dernier

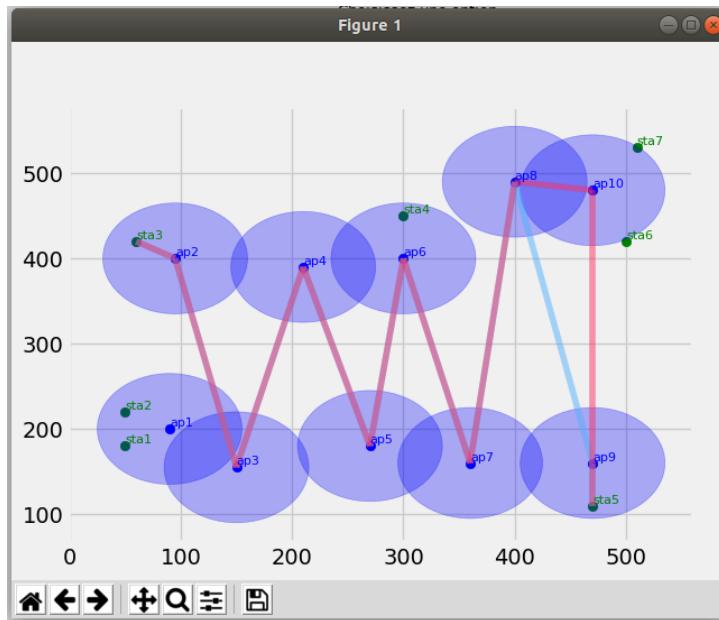


FIGURE 5.8 – Chercher tous les chemins existants(deuxième chemin)

chemin s'afficher (figure 5.9).

2.4 Recherche des chemins possibles selon l'EAR-RP

La fonction `find_all_possible_paths(net, src, dst, aps, stations, energies, path=[])` : est similaire à `find_all_paths`, mais elle inclut une condition supplémentaire pour vérifier si l'énergie du noeud voisin est supérieure à un seuil spécifié (`energy_seuil`). Cela permet de filtrer les chemins qui ne sont pas réalisables en raison de contraintes d'énergie. La fonction utilise la liste `energies` pour vérifier l'énergie restante de chaque noeud.

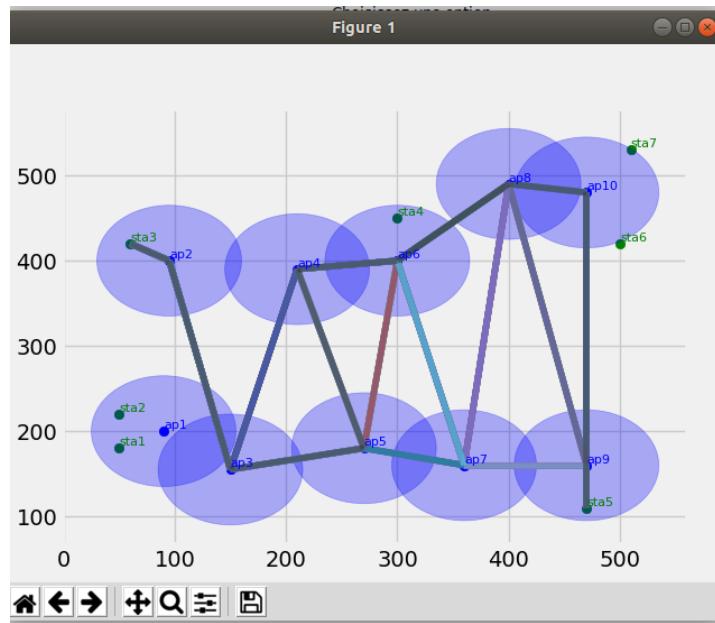


FIGURE 5.9 – Chercher tous les chemins existants(dernier chemin)

Elle renvoie une liste de tous les chemins possibles qui respectent les contraintes d'énergie.

```
def find_all_possible_paths(net, src, dst, aps, stations, energies, path=[]):
    path = path + [src]
    if src == dst:
        return [path]
    paths = []
    for neighbor in neighbors(src, aps, stations):
        if neighbor not in path:
            if energies[neighbor]>energy_seuil:
                new_paths = find_all_possible_paths(net, neighbor, dst, aps, stations,
                                                    path)
                for new_path in new_paths:
                    paths.append(new_path)
    return paths
```

FIGURE 5.10 – chercher tous les chemins possibles selon l'EAR-RP

Nous pouvons apercevoir le résultat de cette fonction à travers notre interface graphique. Il suffit de remplir des zones de texte pour spécifier les noeuds source et destinataire et ensuite il faut cliquer sur le bouton "Afficher les chemins possibles". Ensuite nous pouvons voir les chemins se dessiner progressivement. Le premier chemin qui s'est dessiné pour notre cas (source=sta1, destination=sta4) est visible sur la figure 5.12. Le deuxième chemin peut alors s'afficher(nous avons choisi des couleurs différentes pour chacun des chemins). Le deuxième chemin pour notre cas est visible sur la figure 5.13 A la fin, nous pouvons voir le dernier chemin s'afficher. Le dernier chemin est visible sur la figure 5.14

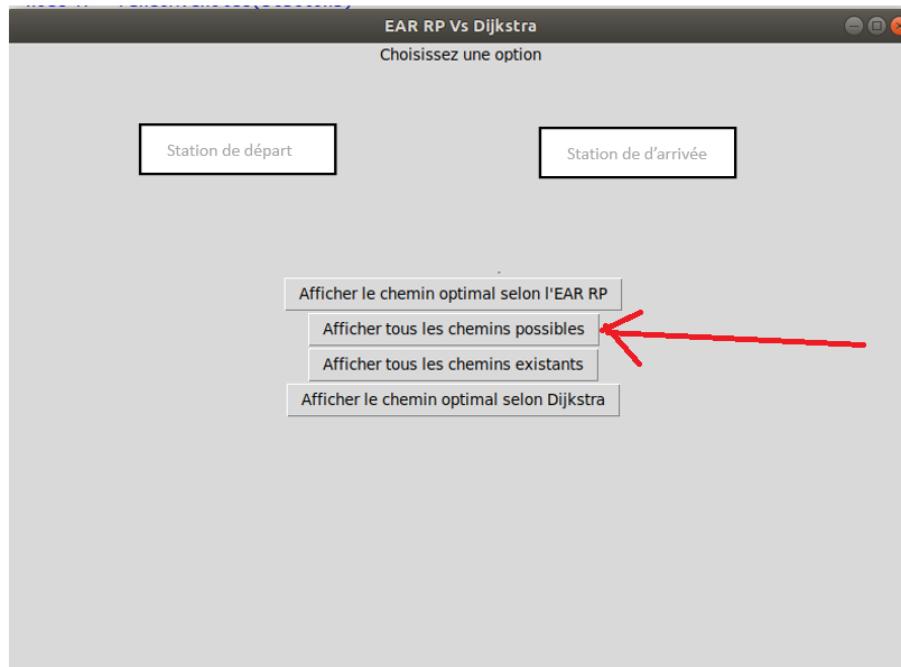


FIGURE 5.11 – Chercher tous les chemins possibles

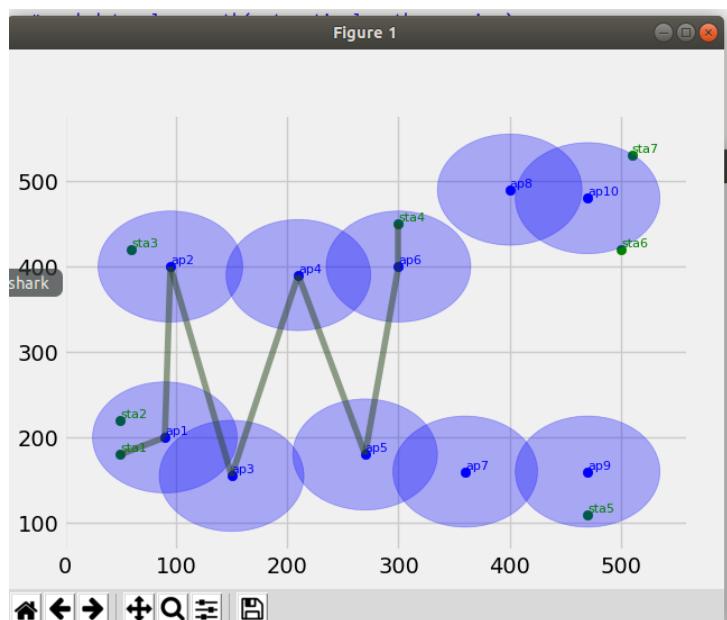


FIGURE 5.12 – Chercher tous les chemins possibles(premier chemin)

2.5 Rechercher le chemin optimal

La fonction `find_optimal_path(possible_paths)` est utilisé pour déterminer le chemin qui minimise l'énergie parmi un ensemble de chemins possibles. Cette fonction est utilisée pour déterminer le chemin optimal selon l'approche EAR RP et Dijkstra. Dans le cas de Dijkstra, elle prend en paramètre l'ensemble des chemins existants (qui sont tous des chemins possibles dans ce cas) et dans le cas de l'EAR RP, elle prend en paramètre l'ensemble des chemins possibles déterminés par `find_all_possible_paths`. On peut voir le résultat sur la figure.

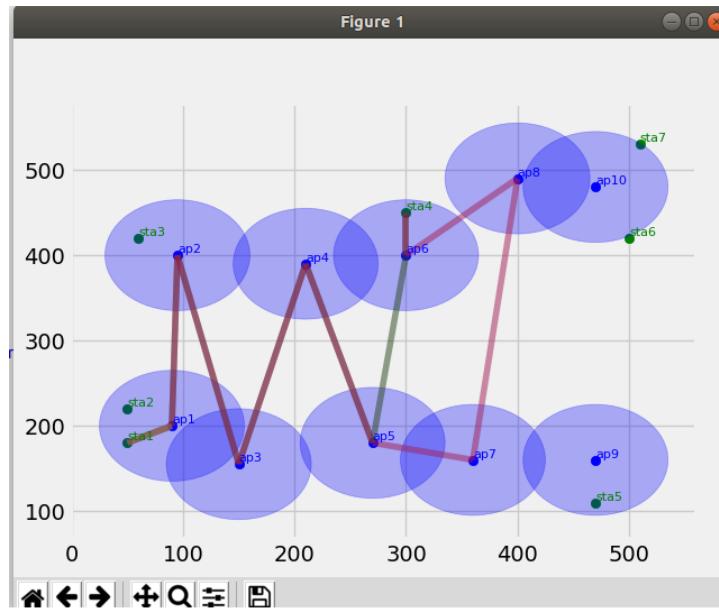


FIGURE 5.13 – Chercher tous les chemins possibles(deuxième chemin)

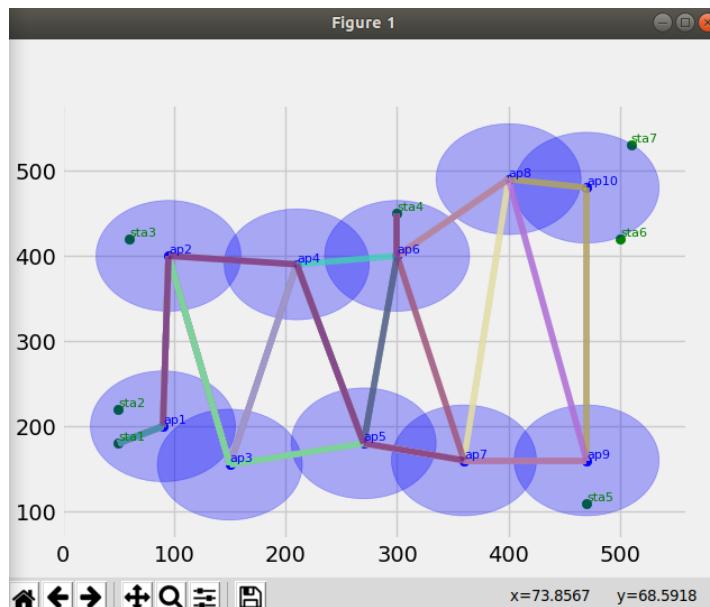


FIGURE 5.14 – Chercher tous les chemins possibles(dernier chemin)

```

def find_optimal_path(possible_paths):
    if len(possible_paths)==0:
        return []
    Ermin=energy_path(possible_paths[0])
    Poptimal=possible_paths[0]
    for Pi in possible_paths:
        if energy_path(Pi)<Ermin:
            Poptimal=Pi
            Ermin=energy_path(Pi)
    return Poptimal

```

FIGURE 5.15 – chercher le chemin optimal

Pour obtenir le chemin optimal, il suffit de choisir soit l'option 1 soit l'option 4 de notre interface. Pour le cas (source=sta3,dest=sta5), nous obtenons la figure 5.16 : En effet, il faut se référer à la topologie (figure

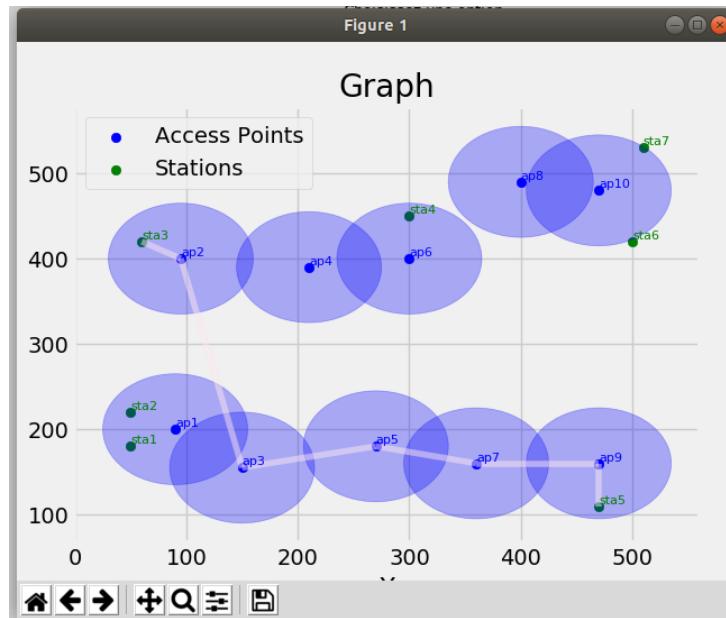


FIGURE 5.16 – chemin optimal selon l’EAR RP

4.4) pour voir les liens qui existent. Par exemple, le chemin sta3-ap2-ap4-ap9-sta5 n’existe pas.

2.6 Affichons les énergies résiduelles des noeuds

La fonction `print_energies(energies)` : Cette fonction affiche les énergies des nœuds du réseau. Elle itère à travers le dictionnaire `energies` et affiche le nom de chaque nœud suivi de son niveau d’énergie. Voici un

```
def print_energies(energies):
    print "Node Energies:"
    for node, energy in energies.iteritems():
        print node.name + ": " + str(energy)
```

FIGURE 5.17 – Fonction pour afficher les énergies des différents noeuds

exemple de résultat que cette fonction affiche (figure 5.18) :

Sur cette figure, nous pouvons voir que l’énergie résiduelle

- Des points d’accès ap1,ap2,...,ap10 sont respectivement de 0, 0, 5, 0, 5, 0, 5, 0, 5, 0
- Des stations sta1, sta2, ..., sta7 sont respectivement de 3.0, 4.5, 5, 5, 5, 3.0, 4.5
- La valeur que l’on peut voir sur la dernière ligne correspond à l’énergie totale dépensée pour arriver à cet état du réseau.

```
Node Energies:  
ap3: 5  
sta2: 4.5  
ap1: 0  
ap7: 5  
ap2: 0  
sta5: 5  
ap6: 0  
sta3: 5  
sta1: 3.0  
sta7: 4.5  
sta4: 5  
sta6: 3.0  
ap8: 0  
ap4: 0  
ap5: 5  
ap10: 0  
ap9: 5  
35.0
```

FIGURE 5.18 – Afficher les énergies des différents noeuds

Conclusion

Cet article met en évidence le problème du routage économique en énergie dans les réseaux définis par logiciel (SD-WCN), qui consiste à optimiser la consommation d'énergie, en particulier dans les zones rurales où les nœuds fonctionnent avec des batteries rechargeables à partir de l'alimentation électrique ou de l'énergie solaire. Pour surmonter ce défi, nous avons proposé EAR-RP, une approche efficace visant à minimiser à la fois la consommation d'énergie totale de la source à la destination et à maximiser l'énergie résiduelle des nœuds intermédiaires le long du chemin en définissant des règles appropriées et en les plaçant dans la table de flux des points d'accès. En bref, le modèle EAR-RP minimise la consommation d'énergie totale et tient compte des contraintes d'énergie résiduelle des nœuds et de capacité de mémoire. Nous avons formalisé le problème sous la forme d'un programme linéaire entier et proposé une heuristique pour le résoudre. Les résultats de l'évaluation montrent que EAR-RP permet de maintenir les nœuds en vie plus longtemps par rapport à l'algorithme du plus court chemin de Dijkstra.

Bibliographie

- [1] S. Andrade-Morelli, E. Ruiz-Sánchez, E. Granell, and J. Lloret. Energy consumption of wireless network access points. *Green Communication and Networking. GreeNets 2012. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 113 :8, 2013.
- [2] D. Bertsimas, B. D. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, pages 464–501, 2011.
- [3] R. R. Fontes, S. Afzal, H. B. S. Brito, M. A. S. Santos, and C. E. Rothenberg. Mininet-wifi : Emulating software-defined wireless networks. *11th International Conference on Network and Service Management (CNSM)*, 2016.
- [4] X. Nguyen, D. Saucez, C. Barakat, and T. Turletti. Rules placement problem in openflow networks : A survey. *IEEE Communications Surveys & Tutorials*, 18(2) :1273–1286, 2016.
- [5] H. M. Noman and M. N. Jasim. Pox controller and open flow performance evaluation in software defined networks (sdn) using mininet emulator. *August 2020 IOP Conference Series Materials Science and Engineering*, 881(1) :012102, 2020.