

Conception d'un Réseau IoT Basé sur SDN

3.1 Introduction

Afin de mettre en oeuvre notre conception, nous avons dédié ce chapitre aux détails de l'évaluation des performances d'un réseau IoT basé sur SDN. Nous commençons par présenter les outils qu'on va utiliser pour atteindre l'objectif de ce projet, puis nous présentons la topologie de réseau et le processus de sa création. Nous terminons ce chapitre par une analyse des résultats en se basant sur différents Paramètres de QoS comme le débit, la gigue, le délai et la perte des paquets.

3.2 Le Contrôleur ONOS :

3.2.1 Aperçu sur ONOS :

ONOS (open network operating system) représente le plan de contrôle dans le SDN, assure la gestion des équipements réseau (ex : switches) et les liens qui les relie, et aussi l'exécution des programmes ou des modules pour fournir les services de communication entre les nœuds et les réseaux adjacents. ONOS offre plusieurs fonctionnalités : des APIs et abstraction, ligne de commande CLI et une interface graphique GUI, allocation des ressources et permissions.

ONOS peut être exécuté comme un système de distribution sur plusieurs serveurs, ce qui permet d'utiliser les ressources de CPU et de la mémoire des serveurs et permet aussi la tolérance contre la panne de serveur.

Le kernel d'ONOS et les services core, et les applications sont programmés en JAVA sous forme

de bundles qui sont chargés dans le conteneur Karaf OSGi. OSGi est un système de composants pour Java qui permet d'installer et d'exécuter des modules de manière dynamique dans une seule JVM (JAVA Virtual Machine) [9].

3.2.2 L'Architecture d'ONOS

L'architecture d'ONOS est composée de 4 éléments principaux [10] :

Le Noyau Distribué

Fournit l'évolutivité, la disponibilité, les performances, onos est déployé sous forme d'un service sur un groupe de serveurs. Et le même logiciel s'exécute dans chaque serveur. Ce qui permet d'assurer la disponibilité en cas d'une panne d'un des serveurs. Les instances d'ONOS fonctionnent ensemble pour créer ce qui apparaît au reste du réseau comme une plate-forme unique.

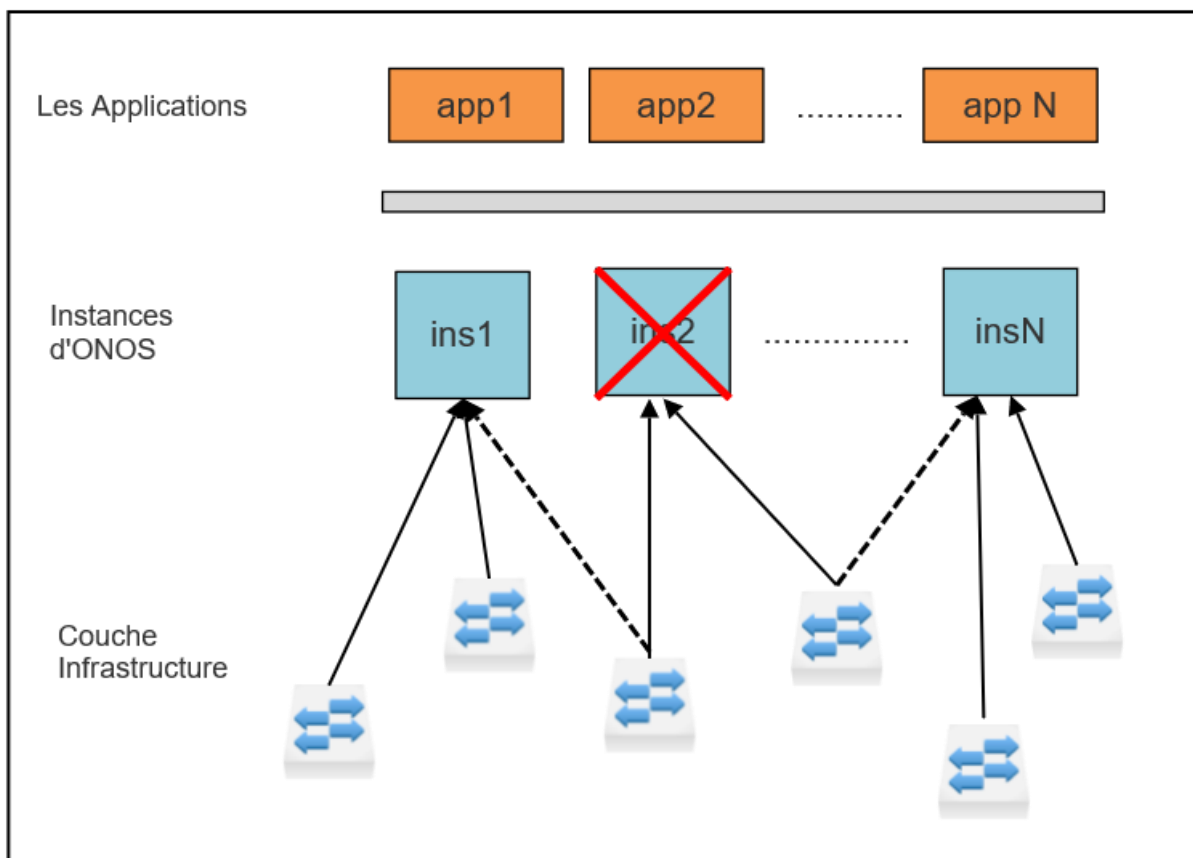


Figure 3.1 – Le Noyau Distribué.

Abstraction Northbound/API

Il existe deux abstractions northbound : le Framework d'intention et la vue de réseau :

Le Framework d'intention permet à une application de demander un service du réseau sans avoir de connaître les détails de service. Cela permet aux opérateurs réseau et les développeurs

d'applications de programmer le réseau a un niveau élevé.

La vue globale du réseau fournit une vue des hosts, switches, liens.

Les Abstractions Southbound

Active les protocoles de niveau bas pour Contrôler les équipements OpenFlow. L'abstraction southbound isole le contrôleur d'ONOS des détails des couches inférieures.

La Modularité des Softwares

La construction des logiciels est importante, le logiciel doit être facile à améliorer, modifier et maintenir. L'équipe ONOS a mis grand soin dans la modularité pour faciliter les développeurs à travailler avec le logiciel.

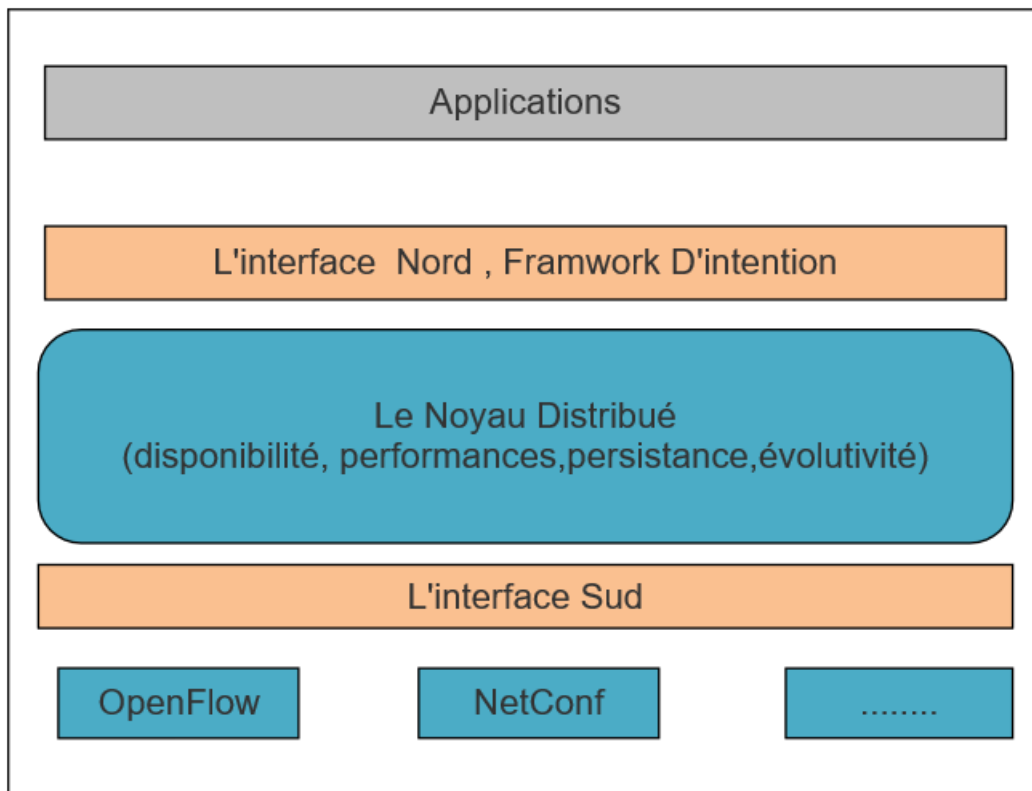


Figure 3.2 –Les Couches d'ONOS.

3.3 Mininet-IoT

3.3.1 Aperçu sur Mininet

Mininet est un émulateur réseau qui permet de créer des topologies réseau virtuelle en utilisant les différents nœuds (switches, contrôleur, hosts) et des liens qui les relient, les switches de mininet supportent OpenFlow pour performer le routage dans le réseau SDN.

Après l'installation de mininet nous pouvons tester une topologie de 2 hosts, 1 switch et 1

contôleur en exécutant "> sudo mn" [11].

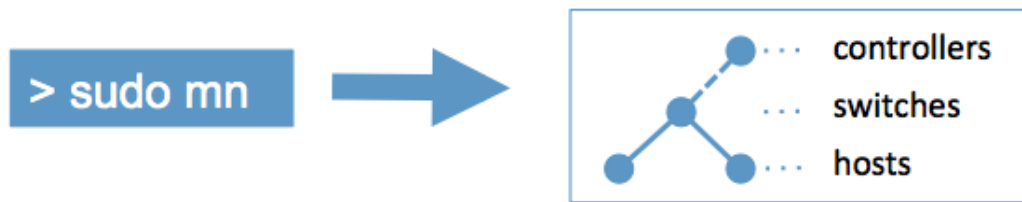


Figure 3.3 –Mininet.

[11]

3.3.2 Mininet-IoT

Mininet-IoT est une extension du codeBase de mininet qui fournit les outils nécessaires pour créer et développer des réseaux IoT. Mininet permet de :

- Fournir une expérience d'essai simple pour développer des applications OpenFlow.
- Inclure une ligne de commande pour consulter et debugger les différents nœuds de réseau.
- Prêt pour l'utiliser sans programmation, mais dans le cas inverse vous pouvez aller plus loin avec PYTHON API.
- Tester des topologies complexes en utilisant l'environnement virtuel.

Sur mininet chaque host a un terminal linux, donc vous pouvez exécuter des commandes linux sur les différents hosts.

Comme il est mentionné au-dessus, il est possible d'utiliser Mininet directement via l'outil en ligne de commande pour créer les différentes topologies SDN. Exemple : la commande suivante permet de créer un switch connecté à 3 hosts(PC) et à un contrôleur bien défini.

```
> sudo mn -t single,3 -c remote,ip=172.0.0.1:6653
```

Ou bien on peut utiliser la PYTHON API, pour programmer notre propre réseau.

3.3.3 Python API

Dans cette implémentation on va utiliser PYTHON pour créer notre propre topologies avec la librairie fournie par python API. Mininet prend en charge les topologies paramétrées. Avec quelques lignes de code Python, vous pouvez créer une topologie flexible qui peut être configurée en fonction des paramètres que vous lui transmettez et réutilisez pour plusieurs expériences.

Remarque: mininet IoT nécessite un kernel 4.18 ou plus.

3.4 Iperf

Iperf est un outil pour mesurer la bande passante et la qualité d'un lien réseau. Le lien dernier est délimité par deux machines sur lesquelles est installé Iperf. Les performances d'un lien peuvent être évaluées comme suit [12] :

- Le Délai : peut être mesuré par la commande de Ping.
- La Gigue (variation de la délai) : par un test Iperf UDP.

- Perte de paquet : mesurée avec un test Iperf UDP.
- Le Débit et Bande passante : peut être mesuré par iperf TCP ou UDP après avoir spécifié la bande passante.
- tester des topologies complexes en utilisant l'environnement virtuelle.

3.5 La Réalisation de la Topologie

3.5.1 Aperçu

Dans cette topologie illustrée dans la figure 3.4 on a 4 groupes chaque groupe contient des capteurs qui forment un réseau maillé et s'échangent des données via le protocole 6LOWPAN, tels que le capteur qui est connecté avec le switch via Ethernet (lien filaire) représente le capteur Edge qui permet de relier le réseau des capteurs avec le réseau filaire. Dans ce test il n'y aura pas de traitement des données (data processing). Mais généralement le capteur Edge qui reçoit les données fait le traitement avant l'envoi au serveur, ou on peut utiliser un serveur de computing qui va faire le traitement et la sauvegarde des données.

Et avec l'application IPv6 réactive forwarding déjà installé sur ONOS les paquets IPv6 des capteurs destinés au serveur seront routés.

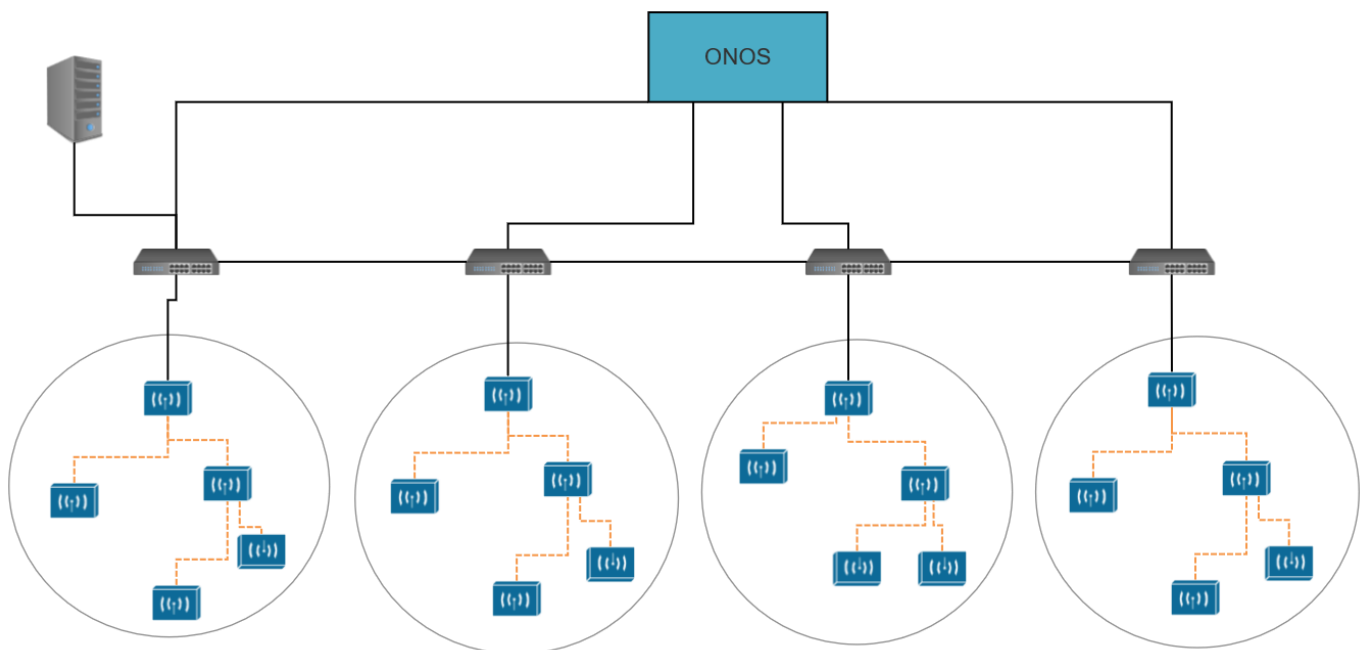


Figure 3.4 –La Topologie Proposée.

3.5.2 Réalisation

On démarre le contrôleur et on entre dans l'interface de commande d'ONOS (ONOS CLI), puis on active l'application de réactif forwarding par la commande "app activate fwd".

Après, il faut activer les fonctionnalités de IPV6 pour le routage des paquets IPV6. Pour cela on exécute les 2 commandes suivantes :

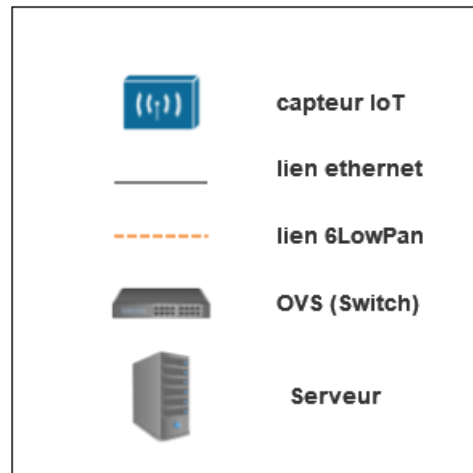


Figure 3.5 –Description.

```
"cfg set org.onosproject.fwd.ReactiveForwarding ipv6Forwarding true"
```

```
"cfg set org.onosproject.fwd.ReactiveForwarding matchIpv6Address true"
```

```
Welcome to Open Network Operating System (ONOS)!

  ONOS

Documentation: wiki.onosproject.org
Tutorials:    tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos> app activate fwd
Activated org.onosproject.fwd
onos> cfg set org.onosproject.fwd.ReactiveForwarding ipv6Forwarding true
onos> cfg set org.onosproject.fwd.ReactiveForwarding matchIpv6Address true
onos>
```

Figure 3.6 –La ligne de command d'ONOS.

Après avoir préparé le contrôleur, on exécute le script Python qui permet de créer la topologie sous de mininet-iot. Dans la phase d'exécution nous consultons l'interface graphique d'ONOS comme il est montré dans la figure 3.7, et voyons les switches OVS (Open V Switch) connecté au contrôleur les capteurs Edge et au serveur, Le deuxième l'hôte connecté au switch 1 est le serveur. On ne voit pas les autres capteurs parce qu'ils ne sont pas connectés directement au switch.

Pour l'évaluation des performances, on a pris 4 scénarios de test, celui qui est dans la topologie de la figure 3.4, les 3 qui restent sont présentés dans la figure 3.9.

Notez bien qu'il nous faut faire un ping entre les capteurs et le serveur pour qu'ils soient détectés par le contrôleur, comme il est montré dans la figure 3.8.

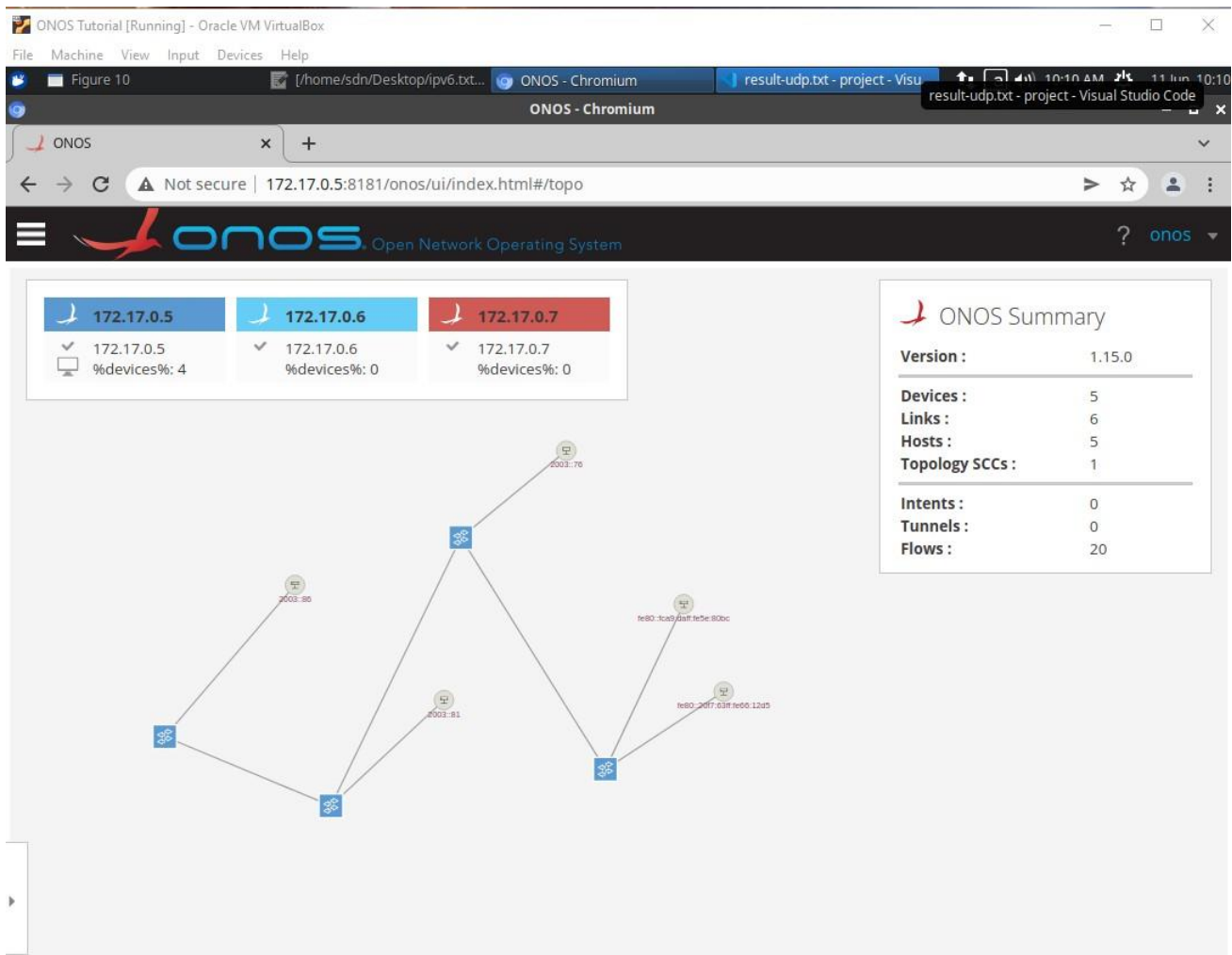


Figure 3.7 –l’Interface Graphique d’ONOS.

3.6 La Qualité de Service

La QoS (Qualité de Service) représente la capacité d’un réseau à gérer la transmission de données de différents flux avec les spécifications requises des paramètres QoS. On peut évaluer la qualité de service avec ces paramètres :

- La Perte de Paquets: représente le pourcentage des paquets qui n’arrive pas à la destination à cause de congestions de réseau qui mène à la suppression des paquets. La perte doit être le plus minimal possible.
- Le Délai: est le temps nécessaire pour qu’un message arrive à la destination. Plus il est proche à zéro plus la QOS est meilleur.
- La Gigue: représente la variation de délai, elle est causé par la congestion de réseau, il doit s’approcher à zéro .
- La Bande Passante: est la quantité d’information maximale que le lien peut supporter dans une période de temps.

Le tableau 3.1 montre les normes Tiphon utilisées dans l’analyse processus. Le débit mesuré

```

102
103 sensor1.cmd("ping -I sensor1-eth1 -c2 2003::100")
104 sensor6.cmd("ping -I sensor6-eth1 -c2 2003::100")
105 sensor11.cmd("ping -I sensor11-eth1 -c2 2003::100")
106 sensor16.cmd("ping -I sensor16-eth1 -c2 2003::100")
107

```

Figure 3.8 –Ping.

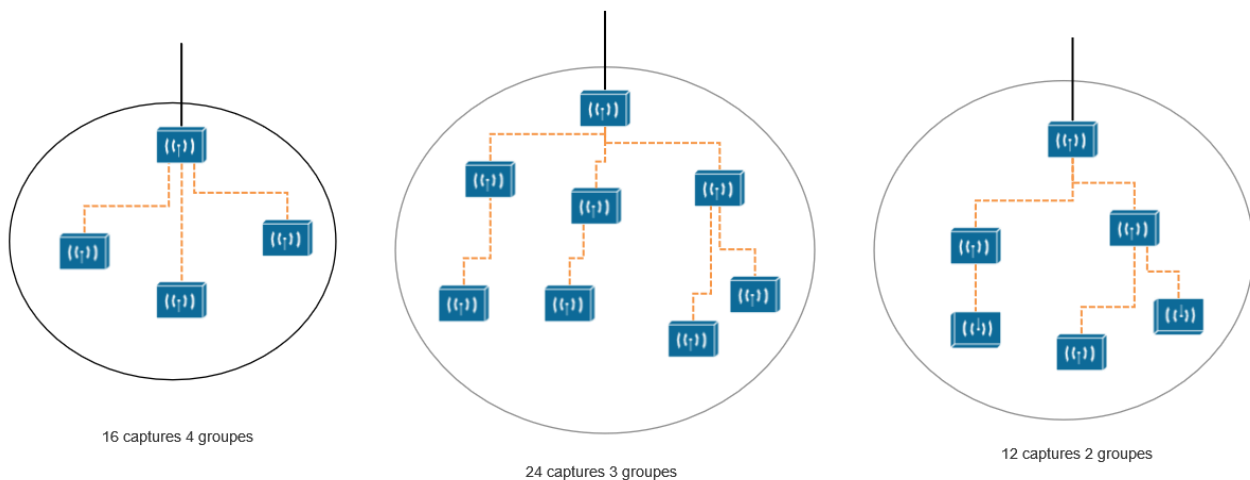


Figure 3.9 –Les Différents Scénarios Testés.

est basé sur le nombre total d'arrivées de paquets réussies observé à destination pendant un intervalle de temps spécifique divisé par la durée de cet intervalle de temps [13].

3.7 Evaluation des performances

comme le montre la figure 3.10, pour que les capteurs qui reçoivent des données, nous devons les définir comme serveurs dans iperf en utilisant l'option -s. On utilise le protocole UDP pour ces tests, -u signifie UDP, -V est pour IPV6 et -p est pour spécifier le numéro de port, -t est juste pour le temps, ce qui signifie que le test ne doit pas dépasser 60 secondes.

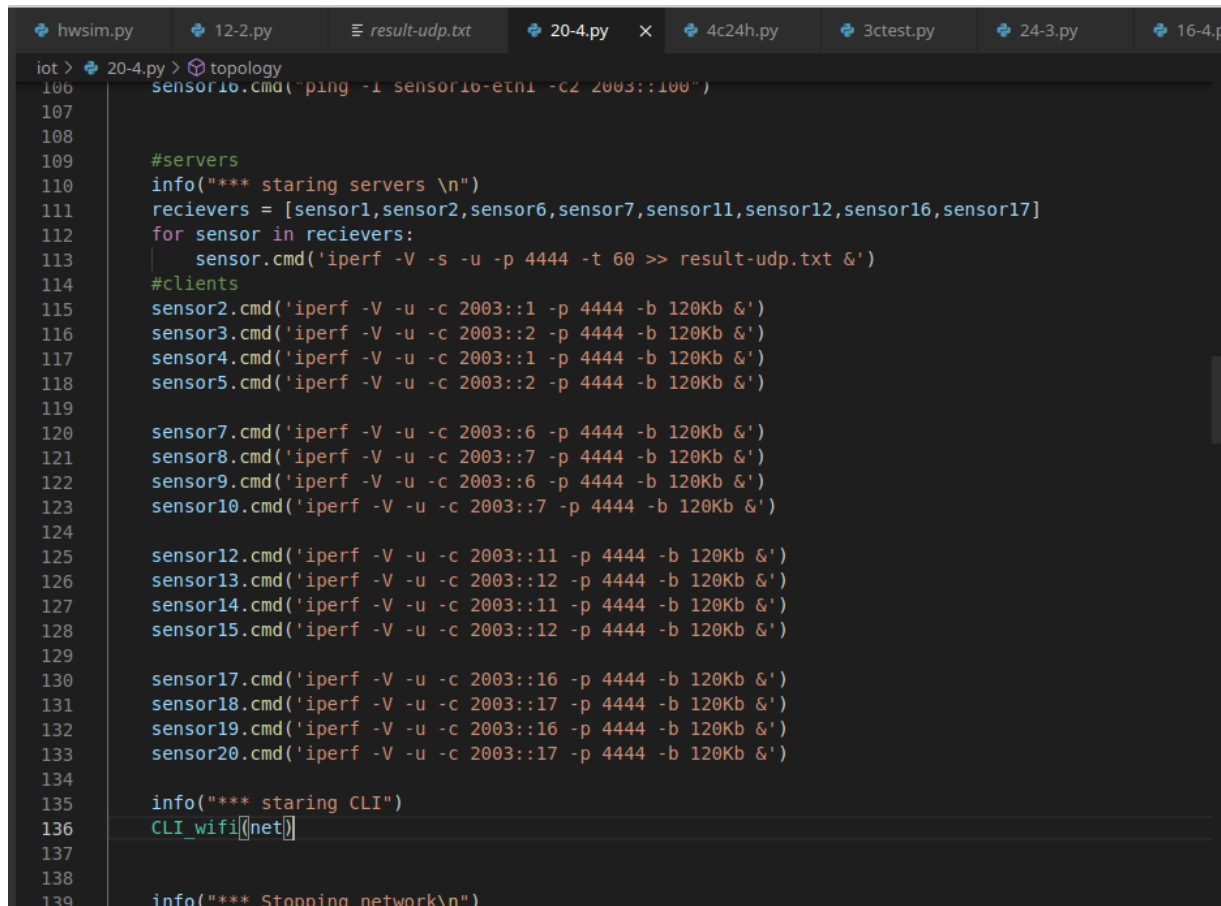
pour les capteurs qui envoient, nous devons les définir en tant que clients dans iperf, nous faisons ça par spécifier l'adresse IP du destinataire dans l'option -c comme indiqué sur la figure 3.10, la bande passante peut être spécifiée avec -b, comme vous pouvez le voir qu'elle est définie sur 120Kbits/s, et les résultats de chaque connexion seront sauvegardés dans le fichier

TABLE 3.1 –Mesure de la QoS [13].

Remarque	Index	Délai (ms)	Gigue (ms)	Perte (%)	Débit (%)
Très Bien	4	150 <	0	0	100
Bien	3	150-300	0-75	3-15	75
Moyen	2	300-450	75-125	15-25	50
Mauvais	1	> 450	> 125	25-100	< 25

result-udp.txt.

nous pouvons voir les résultats de débit, de la gigue et de la perte des paquets dans la figure 3.11, le délai peut être calculer par la commande de ping. on refaire le test plusieurs fois et on prend la moyenne. le scénario de test utilisé est RRUL (Realtime Response Under Load). Ce type de test simule un réseau dans les pires conditions en mettant une charge sur le réseau puis faire la récupération des données. on refaire la même processus avec d'autres scénarios de test dans la figure 3.9, les résultats sont présentés dans les figures 3.12, 3.13, 3.14, 3.15.



```

iot > 20-4.py > topology
106 sensor10.cmd('ping -i sensor10-eth1 -c 2 2003::100')
107
108
109 #servers
110 info("**** starting servers \n")
111 receivers = [sensor1,sensor2,sensor6,sensor7,sensor11,sensor12,sensor16,sensor17]
112 for sensor in receivers:
113     sensor.cmd('iperf -V -s -u -p 4444 -t 60 >> result-udp.txt &')
114 #clients
115 sensor2.cmd('iperf -V -u -c 2003::1 -p 4444 -b 120Kb &')
116 sensor3.cmd('iperf -V -u -c 2003::2 -p 4444 -b 120Kb &')
117 sensor4.cmd('iperf -V -u -c 2003::1 -p 4444 -b 120Kb &')
118 sensor5.cmd('iperf -V -u -c 2003::2 -p 4444 -b 120Kb &')
119
120 sensor7.cmd('iperf -V -u -c 2003::6 -p 4444 -b 120Kb &')
121 sensor8.cmd('iperf -V -u -c 2003::7 -p 4444 -b 120Kb &')
122 sensor9.cmd('iperf -V -u -c 2003::6 -p 4444 -b 120Kb &')
123 sensor10.cmd('iperf -V -u -c 2003::7 -p 4444 -b 120Kb &')
124
125 sensor12.cmd('iperf -V -u -c 2003::11 -p 4444 -b 120Kb &')
126 sensor13.cmd('iperf -V -u -c 2003::12 -p 4444 -b 120Kb &')
127 sensor14.cmd('iperf -V -u -c 2003::11 -p 4444 -b 120Kb &')
128 sensor15.cmd('iperf -V -u -c 2003::12 -p 4444 -b 120Kb &')
129
130 sensor17.cmd('iperf -V -u -c 2003::16 -p 4444 -b 120Kb &')
131 sensor18.cmd('iperf -V -u -c 2003::17 -p 4444 -b 120Kb &')
132 sensor19.cmd('iperf -V -u -c 2003::16 -p 4444 -b 120Kb &')
133 sensor20.cmd('iperf -V -u -c 2003::17 -p 4444 -b 120Kb &')
134
135 info("**** starting CLI")
136 CLI_wifi[net]
137
138
139 info("**** Stopping network\n")
  
```

Figure 3.10 –Tests d'Iperf.

Le Débit

la figure 3.12 montre les résultats de débit, Pour les 4 tests on a attribué 120KB/s de bande passante et nous avons obtenu un débit de 81Kbit/s (68%) et 63Kbit/s (50 %) respectivement. Et pour les 2 autres 65Kbit/s de bande passante et nous avons obtenu un débit de 45Kbits/s (30 %) et 46Kbits/s (36%), alors le débit tombe dans les catégories de moyen et bien.

```

hwsim.py 12-2.py result-udp.txt x 20-4.py 4c24h.py 3ctest.py
result-udp.txt
55 [ 3] local 2003::7 port 4444 connected with 2003::10 port 33191
56 [ 4] local 2003::7 port 4444 connected with 2003::9 port 52529
57 [ 4] local 2003::2 port 4444 connected with 2003::4 port 49067
58 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
59 [ 3] 0.0-14.4 sec    152 KBytes    86.5 Kbits/sec  39.752 ms    0/ 107 (0%)
60 [ 4] 0.0-16.1 sec    152 KBytes    77.0 Kbits/sec  56.195 ms    0/ 107 (0%)
61 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
62 [ 3] 0.0-18.9 sec    152 KBytes    65.5 Kbits/sec  82.569 ms    0/ 107 (0%)
63 [ 4] 0.0-19.4 sec    152 KBytes    64.1 Kbits/sec  86.417 ms    0/ 107 (0%)
64 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
65 [ 3] 0.0-23.1 sec    143 KBytes    50.8 Kbits/sec  139.074 ms    5/ 106 (4.7%)
66 [ 4] 0.0-23.7 sec    139 KBytes    47.9 Kbits/sec  147.179 ms    8/ 106 (7.5%)
67 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
68 [ 3] 0.0-25.6 sec    127 KBytes    40.7 Kbits/sec  187.471 ms   16/ 106 (15%)
69 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
70 [ 3] 0.0-60.3 sec    108 KBytes    14.6 Kbits/sec  292.595 ms    0/ 76 (0%)
71 [ 4] 0.0-60.7 sec    126 KBytes    17.0 Kbits/sec  171.730 ms    0/ 89 (0%)
72 [ 4] 0.0-60.5 sec    108 KBytes    14.6 Kbits/sec  293.425 ms    0/ 76 (0%)
73 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
74 [ 3] 0.0-60.5 sec    109 KBytes    14.8 Kbits/sec  293.620 ms    0/ 76 (0%)
75 [ 3] 0.00-60.54 sec  1 datagrams  received out-of-order
76 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
77 [ 3] 0.0-61.1 sec    112 KBytes    15.0 Kbits/sec  248.576 ms    0/ 79 (0%)
78 [ 4] 0.0-61.3 sec    110 KBytes    14.8 Kbits/sec  257.584 ms    0/ 78 (0%)
79 [ 4] 0.0-62.2 sec    108 KBytes    14.2 Kbits/sec  276.186 ms    1/ 77 (1.3%)
80 [ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
81 [ 3] 0.0-62.9 sec    115 KBytes    14.9 Kbits/sec  224.624 ms    0/ 81 (0%)
82 [ 4] 0.0-62.9 sec    115 KBytes    14.9 Kbits/sec  223.441 ms    0/ 81 (0%)

```

Figure 3.11 –Les Résultats de Test.

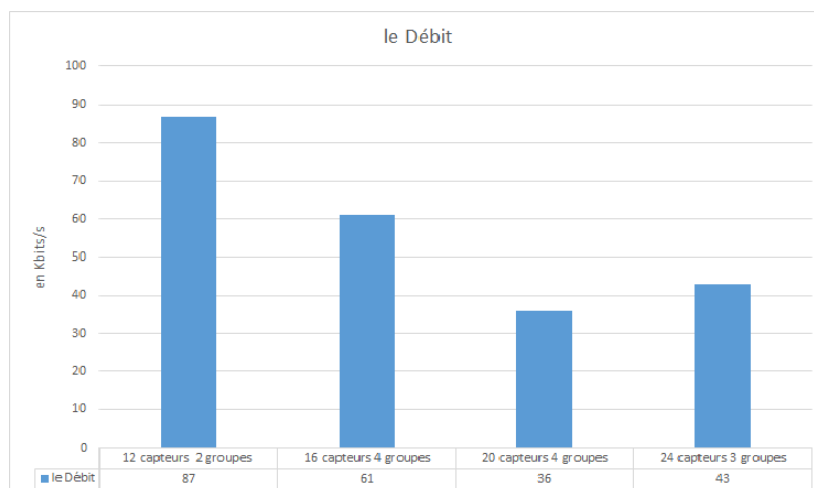


Figure 3.12 –Le Débit.

3.7.1 La Gigue

la figure 3.13 montre les les résultats de la gigue, nous remarquons une relation de corrélation directe entre le nombre de captures et la gigue, Nous avons obtenu 57ms, 63ms, 185ms et 146ms pour les 4 scénarios. Des performances bonnes et moyennes pour 12 capteurs 2 groupes et 16 capteurs 4 groupes respectivement. De mauvaises performances pour les 24 capteurs 3 groupes

et 20 capteurs 4 groupes. Les tests de gigue tombent dans les catégories de bonne et moyenne.

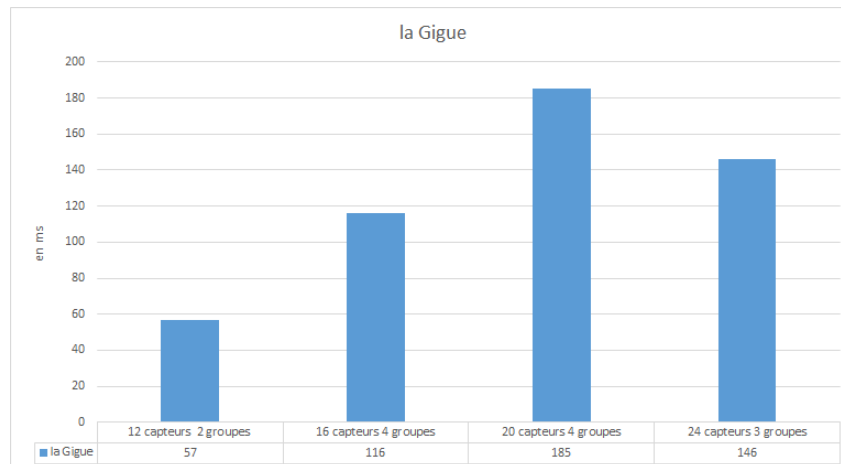


Figure 3.13 –La Gigue.

La Perte

dans la figure 3.14 nous remarquons que la perte augmente en fonction de nombre des capteurs, 0% pour le test de 12 capteurs ce qui est très bien, Et 2%, 2.2% et 4.5% pour les autres ce qui est bien.

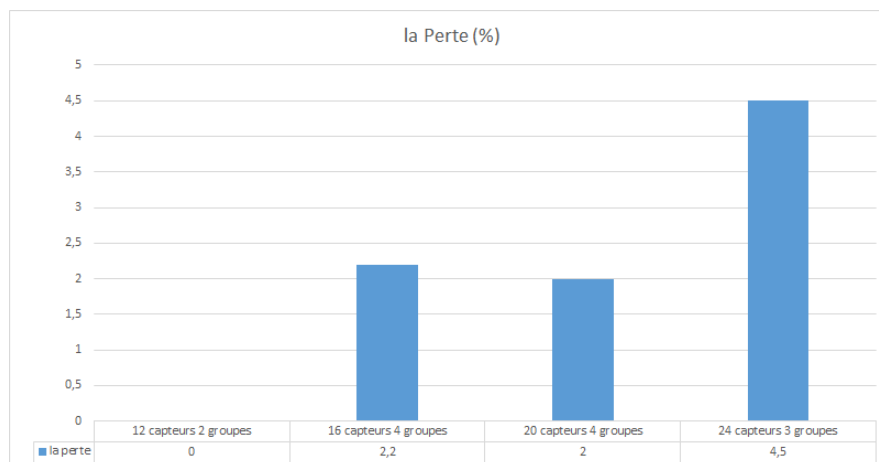


Figure 3.14 –La Perte.

Le délai

la figure 3.15 montre les les résultats de la délai, nous remarquons Encore une relation de corrélation directe entre le nombre de capture et le délai, On a obtenu 11.5ms, 15ms, 19ms et 20ms pour les 4 scénario. Les tests de gigue tombent dans la catégories de bien.

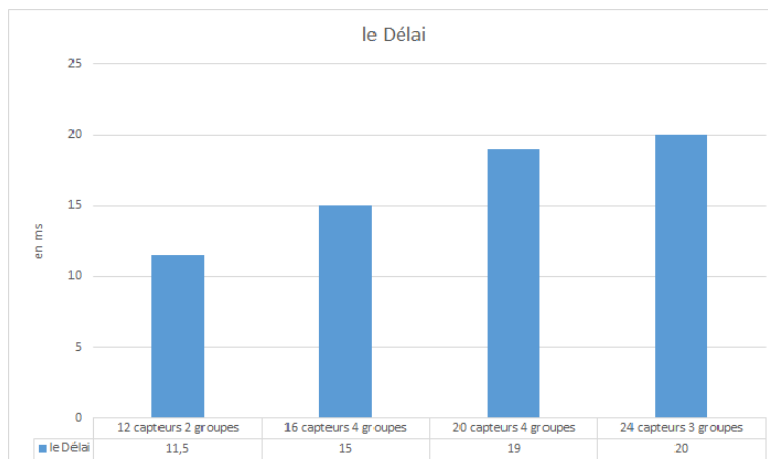


Figure 3.15 –Le Délai

Performances d'Ethernet (Vlaire)

Les performances en réseau Vlaire sont très bien et similaire dans tous les scénarios.

—Le Débit :1 Mbit/s.

—La Gigue :0 - 0,04 ms.

—La Perte :0 %.

—Le Délai :0 - 35 ms.

3.8 Conclusion

Sur la base des résultats des valeurs de débit, délai, gigue et de perte de paquets on peut dire que les performances de réseau 6Lowpan sont moyen-bien avec les meilleur résultats dans le cas de 2 groupes 12 captures. nous avons déduit que Plus le nombre de capture est élevé plus les performances baissent, les 2 derniers tests 3 groupes, 24 captures et 4 groupes, 20 captures ont donné de mauvaises performances de la gigue, Ce qui n'est pas bien pour les applications en temps réel comme la VoIP (Voice over Internet Protocol) par exemple en cas d'un grand réseau qui contient beaucoup de capteurs.