IPV6 for IoT - Practice lab

(Part 1 : IPv6 basics)

Objectives: To practice IPv6 basics (auto-configuration, router advertisements, multicasting, routing) and assess from the experiments some of the benefits and the challenges of using IPv6 in an IoT context.

1. Network settings

Figure 1 describes the network that we consider for this lab. It is composed of a set of end-devices (namely, the end-hosts of room GEI-101) connected to each other via a physical IoT network (emulated here by an Ethernet switch, which can be seen as a radio gateway) and to a Network Gateway (or server, represented here by a Cisco router). The network gateway is then connected to an application server.

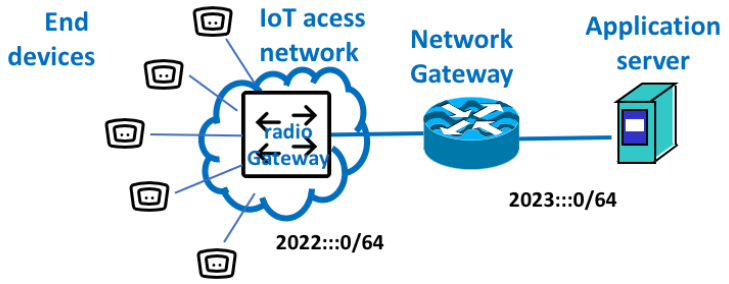


Figure 1.

2. Preliminaries

This first step is to connect the “eth0” of your end-device to the switch, set it up and verify that it can be connected to an IPv6 network by checking the **“/proc/net/if_inet6”** configuration file.

```
bash# ifconfig eth0 up
```

3. Auto-configuration of Link-local addresses

1. First, ensure that the network configuration of network interface “eth0” is blank.

```
bash# ifconfig eth0 down
bash# ifconfig eth0
```

2. Without setting any IP address, turn the network interface “eth0” up as follows

```
bash# ifconfig eth0 up
alternatively
bash# ip link set dev eth0 up
```

check whether “eth0” gets assigned an IPv6 address with

```
bash# ifconfig eth0
bash# ip -6 addr ls dev eth0
```

If so, what type of address is it ? and how was it assigned to eth0 ?

You can also check the subscription of the end-host to these multicast groups with

```
bash#netstat -ng
```

3. In order to study IPv6 procedures when activating interface eth0. Launch tcpdump/“wireshark” by capturing traffic on “*eth0” and filtering on IPv6*

```
bash#tcpdump -i eth0 -n ip6
```

Turn eth0 down, then up while keeping tcpdump running. Analyze the captured data frames and try to identify DAD (Duplicate Address Detection) procedures. Pay attention to the destination addresses that are in use, which are multicast addresses.

4. While still sniffing with wireshark, using the tool “*ping6*” , try to ping one of your neighbors’ IPv6 address. Did it work ? If not, what did you do wrong ? Once you succeed, test the following command

```
bash#ip -6 neigh show
```

To what table from the IPv4 world, this table corresponds to? Identify how IPv6 to MAC address resolution is performed ? Check the destination IPv6 addresses that were used for the resolution ? What is the rationale of using these multicast addresses?

5. While still capturing the traffic, set a new link-local IPv6 address to your node and set it, on purpose, to the IP address of one of your neighbors as follows (you can even ask your neighbour to set an “easy-to-remember” new link-local address (i.e. FE80::1/64) and then set this address to your interface).

```
bash# ifconfig eth0 inet6 add <new_ipv6>/<mask>
or alternatively
bash# ip address add <new_ipv6>/<mask> dev eth0
```

Analyze the captured data frames and check the resulting network configuration of eth0

```
bash# ip -6 addr ls dev eth0
```

Did the DAD succeed ? Confirm it by trying to ping the address of your neighbor.

Remove the duplicate address and check eth0 network configuration.

4. Auto-configuration of unicast global addresses

1. Before configuring the router, check the presence of “router-solicitation” icmpv6 packets sent periodically by your node ?

2. IPv6 forwarding needs to be enabled on the network gateway as follows

```
cisco(config)# ipv6 unicast-routing
```

3. Configure the IP address of gateway’s left-hand network interface as follows

```
cisco(config)# configure terminal
cisco(config)# interface <ifname>
cisco(config-if)# ipv6 enable
cisco(config-f)# ipv6 nd ra interval 30 (set ra
period:30s, not mandatory)
```

```
cisco(config-f)# no shutdown
```

the IPv6 configuration of router's interface can be checked as follows

```
cisco# show ipv6 interface <ifname>
```

4. Check the transmission of "router-advertisement" from the router by capturing the traffic on your node interface?

5. Check that end-hosts are allowed to derive their network configuration from the "router-advertisements" sent by the router by :

```
bash# cat /proc/sys/net/ipv6/conf/eth0/accept_ra
```

the output should be 1.

6. Also, Have a look at the configuration of your node (IPv6 addresses and routing table).

```
bash# ifconfig eth0
bash# route -A inet6
alternatively
bash# ip -6 route
```

Did the "router-advertisement"(s) have an impact on the networking configuration of your node ?

7. Configure the IP address of gateway's left-hand network interface as follows

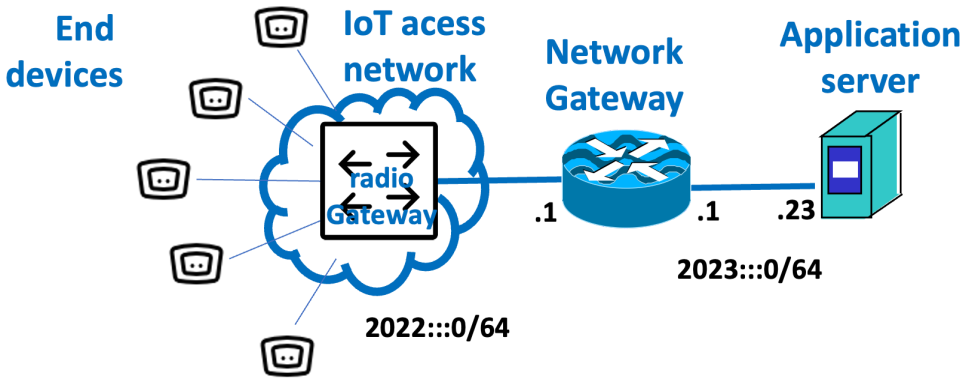
```
cisco(config)# interface <ifname>
cisco(config-f)# ipv6 address <ip>/<mask>
cisco(config-if)# no ipv6 nd ra suppress all
```

8. Check again the network configuration on end devices, including the end hosts' routing table?

9. Check that you can ping the **public** IPv6 address of the router and your neighbors?

5. end-to-end IPv6 connectivity & data distribution with MQTT

Last step is to establish the end-to-end IPv6 connectivity with the application server and set up MQTT. To that end, each row of three end-hosts sets up the following topology composed of two sensors, one application server and an Ipv6 router in between.



1. set up the end-to-end connectivity with the application server.
2. Any application or middleware supporting IPv6 can leverage on this Pv6 connectivity.

Below, we consider using MQTT. The mosquitto MQTT broker as well as its publisher and subscriber clients are installed on all end-devices. On the application server, allow anonymous access to the broker as well as IPv4 and IPv6 accesses, and the broker port number by updating the configuration file of the broker, i.e. "/etc/mosquitto/mosquitto.conf" as follows

```
allow_anonymous true
listener 1883
```

turn the MQTT broker on, with

```
bash# service mosquitto start
```

Then, check that it is running by verifying that the broker is listening on the default port number 1883

```
bash# netstat -an | grep 1883
```

3. you can now use the “mosquitto_sub” and “mosquitto_pub” tools to distribute data between “topic publishers” and “topic consumers” as follows

    - when subscribing to a topic :

      ```
      bash# mosquitto_sub -h <IP address of broker>  -t <topic identifier>
      ```

    - when publishing an instance :

      ```
      bash# mosquitto_pub -h <IP address of broker>  -t <topic identifier> -m <produced instance>
      ```

6. Takeaways regarding the benefits and challenges of adopting IPv6 for IoT

1. Is the address space suited for IoT ?
2. What about human intervention on network configuration of end-devices or things ?
3. What are the requirements induced by IPv6 on the underlying physical network ? Discuss whether these requirements are compatible with IoT networks ?

IPV6 for IoT - Practice lab

(Part 2 : 6LowPAN basics)

Objectives: To practice some aspects of 6LowPAN (mainly, its header compression mechanism) and RPL routing protocol.

1. Network settings

A défaut d'interfaces IEEE 802.15.4 disponibles au sein du département, nous reposerons sur l'émulateur de réseau mininet [1] et plus spécifiquement sur une branche un peu moins connue mininet-wifi [2]. Cette version émule partiellement la pile protocolaire 802.15.4 et certains aspects de 6LowPAN dont la compression d'en-tête IPv6.

Le réseau qui sera émulé est présenté à la figure 1. Dans un premier temps, le protocole de routage RPL sera désactivé pour se concentrer sur la compression d'en-tête 6LowPAN. Dans cette première étape, seules les adresses IPv6 link-local sont en place. Les adresses IPv6 privées sont configurées lorsque le protocole de routage RPL sera activé dans la deuxième étape.

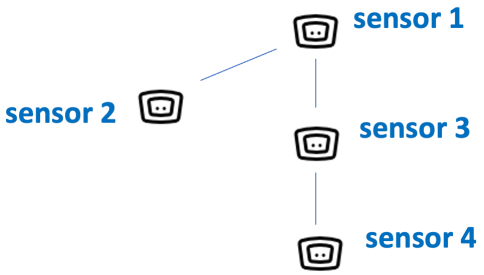


Figure 1. Topologie considérée

2. Preliminaries

Mininet-wifi repose sur une implémentation du protocole de routage RPL dépréciée. Nous utiliserons donc une image de VM préfabriquée qui inclut tous les outils nécessaires dont l'implémentation du protocole RPL.

Connectez vous sous Windows et récupérez l'image de la VM mininet-wifi depuis le lien suivant : Image VM (La taille de l'image est malheureusement importante, le mieux serait de l'installer en local sur la machine de travail ..)

Lancer virtualbox et importez l'image téléchargée (le mot de passe est : *wifi*).

Lancez mininet-wifi avec la topologie suscitée

```
bash# sudo python examples/6LowPan.py
```

A partir de l'invite de commande mininet-Wifi, vous pouvez utiliser les utilitaires classiques en précédant la commande par le nom de la station. Vous trouverez ci-après qcq exemples

```
mininet-wifi> nodes  # liste des noeuds du réseau émulé
mininet-wifi>sensor1 ifconfig # lance la commande ifconfig sur le noeuds sensor1
mininet-wifi>sensor1 xterm & # ouvre un invite de commande sur sensor1
mininet-wifi>sensor1 ping -c1 <@IPv6> # émission depuis sensor1 d'un seul ping
mininet-wifi>sensor1 route -A inet6 # affiche table de routage IPv6 senor1
mininet-wifi>sensor1 iwpan dev sensor1-wpan0 info # récupère les infos relative à l'interface wpan dans sensor1 (pour plus d'option : sensor1 iwpan --help)
```

A noter qu'il est recommandé de lancer une commande de réinitialisation entre deux simulations successives

```
bash# mn -c
```

Vous pouvez rapidement parcourir le fichier "*examples/6LowPan.py"* pour retrouver la topologie réseau de la figure 1.

3. 6LowPAN Header compression

1. Déterminez le plan d'adressage du réseau émulé (adresses IPv6 des différentes noeuds) ?

2. Testez la connectivité IPv6 entre les différents noeuds ? Correspond elle à ce qui est attendu ?

3. Ouvrez un "xterm" sur l'un des noeuds et lancez wireshark et capturez le trafic sur l'interface "sensor-wpan0" puis lancer un ping avec un noeud voisin. Analysez l'encapsulation 6LowPAN pour les trames suivantes :
    a. "router-sollicitation" qui sont typiquement envoyés en multicast !
    b. "neighbor Advertisement", "icmpv6-echo-request" ou "icmpv6-echo-response" qui sont envoyés en unicast.

En analysant le PDU 6LowPAN, identifiez les modes utilisés pour compresser les différents champs de l'en-tête IPv6.

4. Aurait-il été possible de compresser davantage l'en-tête ? Si oui, sous quelle condition ?

4. RPL

Lancez mininet-wifi avec la topologie suscitée avec l'option "-r" qui permet d'activer RPL

```
bash# sudo python examples/6LowPan.py -r
```

1. Déterminez le plan d'adressage du réseau émulé (adresses IPv6 des différentes noeuds) et notamment le plan relatif aux adresses privées ? pourquoi ont-elles été configurées/ajoutées ?
2. Testez la connectivité entre les différents noeuds et constatez ce qu'a permis RPL?
3. En consultant le fichier "*examples/6LowPAN.py",* quel est le noeud racine du DODAG RPL?
4. Consultez les tables de routage des différents noeuds et retrouvez le fonctionnement du protocole RPL et notamment son adéquation au trafic multipoint à point ?
5. Lancez wireshark sur le noeud sensor1 et capturez le trafic sur l'interface "*wpan0"*. Retrouvez le fonctionnement du protocole RPL ?
6. En continuant la capture de trafic au niveau du "*sensor1*", lancez un ping depuis "*sensor4*" à destination de "*sensor2*". Analysez le trafic de routage généré ?

5. Conclusions

1. En quoi consiste la fonction de compression d'entête 6LowPAN ?
2. parmi les différentes fonctionnalités implémentées et assurées par 6LowPAN, lesquelles vous paraissent les plus cruciales pour permettre l'utilisation d'une pile protocolaire basée IPv6 sur un réseau physique IoT contraint ?
3. En vous référant à ce que vous avez observé, pourquoi est ce que le protocole de routage RPL est adapté à du trafic multi-point à point ?

6. Annexe : version dockerisée de Mininet-wifi

Afin de ne pas alourdir les images Ubuntu chargée en séance, une version conteuneurisé de Mininet-Wifi peut être utilisée et complétée par l'installation de qcq outils supplémentaires. Malheureusement, ce conteneur ne dispose pas de l'implémentation du protocole de routage RPL.

Démarrez votre machine en sélectionnant l'image TP-GEI.

Clonez le dépôt Mininet-Wifi comme suit

```
bash# git clone https://github.com/intrig-unicamp/mininet-wifi
```

Un dockerfile est fourni afin de générer le conteneur. Editez-le et modifiez et rajouter l'option -6 pour y inclure l'émulation des noeuds IEEE802.15.4 comme suit

```
bash# cd mininet-wifi
bash# nano Dockerfile
```

Rajouter l'option "6" à la fin de la ligne "RUN util/install.sh -Wlnfv" comme suit "RUN util/install.sh -Wlnfv6"

créez l'image correspondante à l'émulateur mininet-wifi

```
bash# docker build -t mn-wifi:v1 .
```

Vous pouvez vérifier la création de l'image

```
bash# docker images
```

Instancier un conteneur mininet-wifi comme suit

```
bash# docker run -it --rm --privileged --env="DISPLAY"
--env="QT_X11_NO_MITSHM=1" -v /tmp/.X11-unix:/tmp/.X11-unix:rw --net
host -v /sys/:/sys -v /lib/modules:/lib/modules -v
/sys/kernel/debug:/sys/kernel/debug -v /var/run/netns:/var/run/netns mn-wifi:v1
```

sur le terminal du conteneur mininet-wifi, installez les utilitaires ping et wireshark (répondez par l'affirmative aux questions qui vous sont posées)

```
bash# apt update
bash# apt install iputils-ping
bash#  apt install wireshark
```