# Etude Rsx Mesh Wifi sur RPI

## Etude projet existant

### Source

* Projet GitHub: <https://github.com/binnes/WiFiMeshRaspberryPi> (*Projet plus maintenu*)
* Wiki protocol Batman: <https://en.wikipedia.org/wiki/B.A.T.M.A.N>.
* Page de Batman: <https://www.open-mesh.org/projects/open-mesh/wikia>
* IBSS /Ad Hoc: <https://www.geeksforgeeks.org/introduction-of-independent-basic-service-set-ibss/>
  + Explication des cas d’usage d’un réseau Ad-Hoc
* IBSS/Ad Hoc mais aussi les autres modes wifi
  + <https://pressbooks.pub/wifi/chapter/modes-de-fonctionnement/>
  + <https://www.tala-informatique.fr/wiki/images/f/f7/WiFi-802.11.pdf>
* OpenVPN création de serveur VPN sur RPI
  + <https://openvpn.net/vpn-server-resources/install-openvpn-access-server-on-raspberry-pi/> tuto fait par openvpn mais passe par leur hébergement
  + <https://raspberrytips.fr/installer-serveur-vpn-raspberry-pi/> tuto fait par des tips rpi mais utilisation d’un script d’installation peu fiable
  + <https://www.clubic.com/raspberry-pi/dossier-365094-transformez-un-raspberry-pi-en-serveur-vpn-avec-pivpn.html> installation assisté avec PiVPN
* Conseil pour sécuriser les RPI
  + <https://raspberrytips.fr/securiser-raspberry-pi/> conseille assez basique pour sécuriser une RPI
  + <https://adamtheautomator.com/iptables-rules/> Firewall basics

### But

Créer un réseau mesh avec RPI: possibilité d'extension + retransmission des données capteurs + connection d'appareils n’appartenant pas au rsx msh

Apprentissage: rsx mesh haut niv + apprendre diff entre “routing & bridging network traffic” + comm entre rsx & cloud de manière sécurisée

### Prérequis

* minimum 2x RPI (3 ou 4 B(+))
* pc, wifi etc… matos basique
* une connection à un cloud (*vm pour les tests ça devrait être ok*)

### Les 2 étapes

1. setup rsx mesh (2 RPI). Connection WIFI, connection rsx local (Gateway router connecté rsx local + autorisé pc externe (non-mesh) co au rsx mesh (Bridge node connecter à [Ad-Hoc](https://en.wikipedia.org/wiki/Wireless_ad_hoc_network)
   1. attention perte de connection à certaines parties
   2. [part 1 connexion Ethernet](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/README.md)
   3. [part 2 connexion wifi](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part2/README.md)
2. [connection sensor](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part3/README.md)

### Connection rsx wifi

#### Gateway

Connexion possible en wifi avec une seconde interface possible pour faire le lien rsx mesh et extérieur. Cependant cela nécessite une clé USB avec un module wifi. *Je pense qu'on va favoriser l’utilisation du câble Ethernet pour le Gateway (voir* [*Part 1 connexion Ethernet Gateway*](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/ROUTE.md)*).*

Différence entre Bridge et Gateway

* Bridge établie la connexion avec Internet
  + Couche rsx niv2
  + Sert juste de pont donc partage la même tranche @ ip que le réseaux extérieur au mesh
* Gateway (**on opte pour cette solution**)
  + Gateway sert de NAT & DHCP donc le réseau local ne peut voir le réseau intérieur par contre les nodes du rsx mesh peuvent voir les pc du réseaux local
  + Principe des réseaux wifi qu’on a chez nous (tranche d’adresse IP etc)
  + couche rsx niv3

#### Node

* norme[802.11s](https://fr.wikipedia.org/wiki/IEEE_802.11s)
* outils [batman-adv](https://www.open-mesh.org/projects/open-mesh/wiki) (fait déjà parti du noyau Linux) *utile à la création de rsx mesh*
* prend contrôle interf **wlan0** puis créé interf **bat0** (*permet de communiquer avec rsx mesh*
* Utilisation d’une image **Raspbian Buster Lite version** (site)

**Configuration de RPI**

**## Flasher l'image**

```bash

sudo apt install rpi-imager

```

**## Logs**

hostname : pi

Mot de passe : tprli

**## Conf de l’interface**

On peut remplacer le channel par un numéro de channel

Ici nous sommes sur le channel **1 (Europe)**

**## Mise en place du noeud**

[**https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/PIMESH**](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/PIMESH)**.**

**md#setup-batman-adv**

[Documentation](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/PIMESH.md#setup-batman-adv)

**## Mise en place du gateway**

Faire le tuto du dessus puis faire celui [là](https://github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/ROUTE.md#creating-the-gateway)

**## Remarque**

- Commande qui ne fonctionne pas sur le gateway

```bash

sudo apt-get update && sudo apt-get upgrade -y

```

### ChatGPT WPA3 & BATMAN

Integrating WPA3 security with a BATMAN mesh network involves addressing security at both the link layer (BATMAN protocol) and the network layer (WPA3). While BATMAN focuses on routing and mesh communication, WPA3 provides encryption and authentication for the WiFi network. Below is a general guide on how you might approach deploying WPA3 in a BATMAN mesh network:

BATMAN Mesh Network Setup:

Set up your BATMAN mesh network as you normally would. Ensure that nodes can communicate with each other through the mesh.

WiFi Configuration:

Configure the WiFi interfaces on your mesh nodes with WPA3 security. This involves setting up encryption and authentication parameters.

Edit the network configuration files on each node. For example, on a Linux-based system with wpa\_supplicant, you might have a configuration file (e.g., /etc/wpa\_supplicant.conf) where you define WPA3 parameters.

Example wpa\_supplicant.conf snippet for WPA3:

plaintext

Copy code

network={

ssid="YourMeshSSID"

key\_mgmt=SAE

proto=RSN

pairwise=CCMP

group=CCMP

psk="YourMeshPassword"

}

Adjust the settings based on your security requirements.

Mesh Node Authentication:

Ensure that each BATMAN mesh node is authenticated and authorized to join the network.

This may involve configuring BATMAN to use authentication mechanisms or ensuring that only authorized nodes have the necessary credentials for WPA3.

Secure Mesh Routing (Optional):

BATMAN itself does not provide encryption for the mesh traffic. If securing the mesh communication is a concern, consider using additional mechanisms like VPNs (Virtual Private Networks) to secure the mesh routing traffic.

Key Distribution:

If your BATMAN mesh network spans multiple hops, consider how WPA3 keys will be distributed securely across the mesh.

Mechanisms like pre-shared keys (PSK) might be used, or you might explore more advanced key management solutions.

Node Configuration and Updates:

Ensure that all mesh nodes are correctly configured with the WPA3 settings. Automated configuration tools or scripts might be useful for large deployments.

Testing and Monitoring:

Test the network thoroughly to ensure that both BATMAN mesh communication and WPA3 security are functioning as expected.

Implement monitoring solutions to detect and respond to security incidents.

Documentation and Training:

Document the configuration settings and security policies applied to your BATMAN mesh network with WPA3.

Train administrators and users on security best practices.

Keep in mind that security is a complex topic, and deploying WPA3 in a BATMAN mesh network requires a thorough understanding of both technologies. Additionally, stay informed about updates and security patches for the software components involved.

### ChatGPT Technologie pour authentification dans un réseau mesh avec Batman

WPA3 (Wi-Fi Protected Access 3):

WPA3 is a security protocol that addresses authentication and encryption for WiFi networks.

It enhances security over WPA2 and can be used in conjunction with WiFi mesh networks to provide secure authentication.

802.1X (Port-Based Network Access Control):

IEEE 802.1X is a standard for port-based network access control that can be used to provide authentication for devices connecting to a network, including mesh nodes.

EAP (Extensible Authentication Protocol):

EAP is an authentication framework that supports various authentication methods.

It is commonly used in enterprise WiFi deployments and can be applied to WiFi mesh networks for authentication.

Pre-Shared Key (PSK):

PSK is a simple form of authentication where devices use a shared secret passphrase to authenticate and establish a secure connection.

While not as secure as more advanced methods, it can still be effective in certain scenarios.

Certificate-Based Authentication:

Certificate-based authentication involves the use of digital certificates to authenticate devices on the network.

This method is often used in enterprise environments and can be applied to WiFi mesh networks.

### ChatGPT VPN dans rsx BATMAN

Installing a VPN (Virtual Private Network) on a BATMAN mesh network involves securing the mesh communication by encapsulating it within a VPN tunnel. Below are general steps for setting up a VPN on a BATMAN-adv mesh network. In this example, we'll use OpenVPN as the VPN software, but you can adapt the steps based on your preferred VPN solution.

\*\*Note:\*\* Ensure you have administrative access to the devices in your mesh network, and that they are capable of running both BATMAN-adv and the chosen VPN software.

1. \*\*Install BATMAN-adv:\*\*

- Set up and configure BATMAN-adv on each node in your mesh network. Follow the BATMAN-adv documentation for your specific platform.

2. \*\*Install OpenVPN:\*\*

- Install OpenVPN on each node in your mesh network. The installation process may vary based on your operating system. On Linux, you might use package managers like `apt`, `yum`, or others.

3. \*\*Generate Certificates and Keys:\*\*

- Use OpenVPN's `easyrsa` or a similar tool to generate the necessary certificates and keys for your OpenVPN setup.

- Create a Certificate Authority (CA), server certificate, and client certificates for each mesh node.

4. \*\*Configure OpenVPN Server:\*\*

- Set up an OpenVPN server on one of the mesh nodes. Create a configuration file (e.g., `server.conf`) with appropriate settings:

```conf

port 1194

proto udp

dev tun

ca ca.crt

cert server.crt

key server.key

dh dh2048.pem

server 10.8.0.0 255.255.255.0

push "route 10.0.0.0 255.255.255.0"

client-to-client

duplicate-cn

comp-lzo

persist-key

persist-tun

```

- Customize the configuration based on your network and security requirements.

5. \*\*Configure OpenVPN Clients:\*\*

- On each mesh node, create a client configuration file (e.g., `client.conf`):

```conf

client

dev tun

proto udp

remote <server-ip> 1194

resolv-retry infinite

nobind

persist-key

persist-tun

ca ca.crt

cert client.crt

key client.key

comp-lzo

```

- Replace `<server-ip>` with the IP address of the OpenVPN server.

6. \*\*Start OpenVPN:\*\*

- Start the OpenVPN server on the designated server node:

```bash

sudo openvpn --config server.conf

```

- Start OpenVPN clients on the other mesh nodes:

```bash

sudo openvpn --config client.conf

```

7. \*\*BATMAN-adv and Routing:\*\*

- Ensure that BATMAN-adv is operational on all nodes and that they can communicate with each other.

- Adjust routing configurations to ensure that mesh traffic is directed through the VPN tunnel.

8. \*\*Testing:\*\*

- Test connectivity between nodes to ensure that mesh traffic is securely routed through the VPN tunnel.

9. \*\*Security Considerations:\*\*

- Consider additional security measures, such as firewall rules, to restrict unauthorized access to the VPN and mesh network.

Keep in mind that this is a basic guide, and the actual steps may vary based on your specific network setup, operating systems, and VPN solution. Additionally, ensure that you comply with relevant legal and security policies when implementing a VPN on your mesh network.

Install OpenVPN

To install and configure OpenVPN on a Linux-based system, you can follow these general steps. Keep in mind that the exact steps might vary depending on your Linux distribution. Here, I'll provide instructions for Debian/Ubuntu systems.

Install OpenVPN:

bash

Copy code

sudo apt update

sudo apt install openvpn

Configuration:

Create Certificates and Keys:

OpenVPN uses certificates and keys for authentication. You can use the EasyRSA script provided by OpenVPN to generate them.

bash

Copy code

sudo mkdir /etc/openvpn/easy-rsa

sudo cp -r /usr/share/easy-rsa/\* /etc/openvpn/easy-rsa/

cd /etc/openvpn/easy-rsa

sudo ./easyrsa init-pki

sudo ./easyrsa build-ca

sudo ./easyrsa gen-dh

sudo ./easyrsa gen-req server nopass

sudo ./easyrsa sign server

Follow the on-screen prompts to provide information for the certificate.

Generate Client Certificates:

Repeat the following commands for each client.

bash

Copy code

cd /etc/openvpn/easy-rsa

sudo ./easyrsa gen-req client1 nopass

sudo ./easyrsa sign client1

Create OpenVPN Server Configuration File:

Create a configuration file for the OpenVPN server. You can use a basic template and adjust it as needed.

bash

Copy code

sudo nano /etc/openvpn/server.conf

Example server.conf:

conf

Copy code

port 1194

proto udp

dev tun

ca /etc/openvpn/easy-rsa/pki/ca.crt

cert /etc/openvpn/easy-rsa/pki/issued/server.crt

key /etc/openvpn/easy-rsa/pki/private/server.key

dh /etc/openvpn/easy-rsa/pki/dh.pem

server 10.8.0.0 255.255.255.0

push "redirect-gateway def1 bypass-dhcp"

push "dhcp-option DNS 8.8.8.8"

push "dhcp-option DNS 8.8.4.4"

comp-lzo

user nobody

group nogroup

persist-key

persist-tun

status /var/log/openvpn/openvpn-status.log

verb 3

Save the file.

Create OpenVPN Service File:

Create a service file for OpenVPN.

bash

Copy code

sudo nano /etc/systemd/system/openvpn.service

Example openvpn.service:

ini

Copy code

[Unit]

Description=OpenVPN service

[Service]

Type=oneshot

ExecStart=/etc/openvpn/setup.sh

ExecReload=/bin/true

WorkingDirectory=/etc/openvpn

RemainAfterExit=yes

[Install]

WantedBy=multi-user.target

Save the file.

Create setup.sh Script:

Create a setup script for OpenVPN.

bash

Copy code

sudo nano /etc/openvpn/setup.sh

Example setup.sh:

bash

Copy code

#!/bin/bash

iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE

systemctl start openvpn

Save the file and make it executable:

bash

Copy code

sudo chmod +x /etc/openvpn/setup.sh

Enable and Start OpenVPN:

bash

Copy code

sudo systemctl enable openvpn

sudo systemctl start openvpn

Configure IP Forwarding:

Enable IP forwarding to allow traffic to be forwarded through the VPN.

bash

Copy code

sudo sysctl -w net.ipv4.ip\_forward=1

Make it persistent:

bash

Copy code

echo "net.ipv4.ip\_forward=1" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p

Client Configuration:

For each client, you'll need to create a client configuration file and provide the client certificate and key.

Create a client configuration file (e.g., client1.ovpn):

conf

Copy code

client

dev tun

proto udp

remote [your\_server\_ip] 1194

resolv-retry infinite

nobind

persist-key

persist-tun

ca ca.crt

cert client1.crt

key client1.key

comp-lzo

verb 3

Replace [your\_server\_ip] with the public IP address or hostname of your OpenVPN server.

Transfer the client configuration file (client1.ovpn), the client certificate (client1.crt), and the client key (client1.key) to the client device.

Connect to the OpenVPN server using the OpenVPN client software, specifying the client configuration file.

Now, your OpenVPN server should be up and running. Adjust the configuration files as needed for your specific requirements and security considerations. This is a basic setup, and depending on your use case, you might want to implement additional security measures, such as firewall rules, certificate revocation lists, etc.

### Communication Noeud < ---- > Gateway

+---------------------+

| Application |

+---------------------+

| Transport |

+---------------------+

| Réseau |

+---------------------+

| Liaison |

+---------------------+

| Physique |

+---------------------+

* SFTP (couche Transport) : transfère de fichier de manière sécurisée. Utilise SSH qui lui même utilise TCP
* Coap (couche Transport): Utilise UDP

| Protocoles Optimisés pour les Réseaux Mesh :  B.A.T.M.A.N. Advanced (BATMAN-ADV) File Transfer: Si vous utilisez déjà le protocole BATMAN pour le routage dans votre réseau mesh, vous pourriez explorer des solutions personnalisées qui tirent parti de BATMAN-ADV pour le transfert de fichiers.  Protocoles Légers et Spécifiques à l'IoT :  MQTT-SN (MQTT for Sensor Networks): Conçu pour les réseaux de capteurs, MQTT-SN est une version légère de MQTT (Message Queuing Telemetry Transport) qui peut être adaptée pour le transfert de fichiers.  CoAP (Constrained Application Protocol): CoAP, comme mentionné précédemment, est un protocole léger conçu pour les appareils avec des ressources limitées. |
| --- |

| MQTT-SN (MQTT for Sensor Networks) est une version du protocole MQTT (Message Queuing Telemetry Transport) conçue spécifiquement pour les réseaux de capteurs et les environnements IoT (Internet of Things) où les dispositifs peuvent avoir des contraintes sévères en termes de ressources. MQTT-SN étend MQTT pour répondre aux exigences spécifiques des réseaux de capteurs et des déploiements IoT avec des appareils à faible puissance et des réseaux à faible bande passante.  Principales fonctionnalités de MQTT-SN :  Protocole Léger : MQTT-SN est conçu pour être plus léger que MQTT, adapté aux appareils avec des ressources limitées tels que les capteurs, les actionneurs et d'autres dispositifs IoT.  Support des Réseaux de Capteurs : Contrairement à MQTT qui est conçu pour les réseaux TCP/IP, MQTT-SN peut fonctionner sur divers types de réseaux de capteurs, y compris les réseaux radio basse puissance (Low-Power Wide-Area Networks - LPWAN), les réseaux mesh, etc.  Découverte Automatique : MQTT-SN prend en charge la découverte automatique des brokers MQTT-SN dans le réseau. Cela permet aux dispositifs de trouver dynamiquement le broker auquel ils doivent se connecter.  Gestion de la Connexion : MQTT-SN utilise un mécanisme de gestion de connexion qui permet aux dispositifs de se connecter et de se déconnecter dynamiquement du réseau tout en maintenant leur identité et leurs abonnements.  QoS (Quality of Service) : MQTT-SN prend en charge plusieurs niveaux de qualité de service pour la livraison des messages, similaire à MQTT. Cela permet de s'adapter aux différentes conditions du réseau et aux exigences de fiabilité.  Gestion de l'Énergie : Les fonctionnalités de MQTT-SN sont conçues pour minimiser la consommation d'énergie des dispositifs, ce qui le rend approprié pour les applications IoT alimentées par batterie.  Topiques MQTT : MQTT-SN utilise également le concept de topiques (topics) pour organiser les données et les commandes, facilitant la publication et la souscription d'informations spécifiques.  Flexibilité du Réseau : MQTT-SN est conçu pour être adaptatif aux différents types de réseaux de capteurs et de dispositifs IoT, offrant ainsi une certaine flexibilité dans le choix de la technologie réseau sous-jacente.  En résumé, MQTT-SN offre une version allégée de MQTT, adaptée aux contraintes des réseaux de capteurs et des environnements IoT où les ressources sont limitées. Son objectif est de permettre une communication efficace et fiable entre des dispositifs hétérogènes au sein d'un réseau. |
| --- |