Sniffer Réseau

En vue de l’obtention du diplôme technique de qualification en informatique

Titulaire : M. Ben Sellam

*Table des Matières*

[1. Visibilité sur le LAN : Pourquoi et Comment Monitorer son Trafic 3](#_Toc197031872)

[2. Contexte implémentation : 3](#_Toc197031873)

[3. Cahier des charges : 3](#_Toc197031874)

[4. Topologie : 3](#_Toc197031875)

[5. Explications concert : 3](#_Toc197031876)

[4.1. Mise en place de la base de données 3](#_Toc197031877)

[4.2. Développement de l’interface web 3](#_Toc197031878)

[4.3. Capture et analyse des trames réseau 3](#_Toc197031879)

[6. Sécurité et gestion des accès 3](#_Toc197031880)

[7. Difficultés Rencontrées et Solutions 3](#_Toc197031881)

[7.1. Problème de Droits Root pour la Capture de Paquets : 3](#_Toc197031882)

[7.2. Performance de la Base de Données avec un Grand Volume de Paquets : 3](#_Toc197031883)

[7.3. Sécurisation de l’Interface Web : 3](#_Toc197031884)

[7.4. Difficultés liées à la Mise en Place de Flask, HTML, CSS, JS : 3](#_Toc197031885)

[8. perspectives amelioration 3](#_Toc197031886)

[9. Conclusion 3](#_Toc197031887)

[10. Annexes 3](#_Toc197031888)

# Visibilité sur le LAN : Pourquoi et Comment Monitorer son Trafic

Mon Travail de Fin d'Études porte sur la conception et la réalisation d'une solution légère et économique de **surveillance et d'analyse de trafic réseau** au sein d'un environnement local (LAN).

L'objectif principal de ce projet est de fournir aux **petites entreprises et aux particuliers** un outil simple pour :

1. **Tester la robustesse de leur infrastructure réseau.**
2. **Identifier les problèmes** de performance ou de configuration.
3. **Détecter de potentielles brèches de sécurité** ou des activités suspectes.

Pour atteindre cet objectif, j'ai développé un **sniffer réseau** basé sur un **Raspberry Pi 4**. Le choix de cette plateforme matérielle a été motivé par son faible coût, sa petite taille et sa capacité à fonctionner de manière autonome en continu.

La solution technique s'articule autour de plusieurs composants clés :

* **Capture en Temps Réel :** J'utilise la librairie Python **Scapy** pour intercepter et analyser les paquets de données qui transitent par le réseau.
* **Stockage des Données :** Les informations pertinentes extraites des paquets (comme les adresses IP source/destination, les protocoles, les tailles) sont stockées dans une base de données légère, **SQLite**.
* **Interface Web :** J'ai développé une interface utilisateur accessible via un navigateur web en utilisant le framework Python **Flask**. Cette interface permet de visualiser de manière conviviale les données de trafic capturées, facilitant ainsi l'analyse pour l'utilisateur.
* **Sécurité :** L'accès à cette interface web est protégé par un système d'authentification simple, et la mise en place de HTTPS est recommandée pour sécuriser les échanges.

Au cours du projet, j'ai effectué des **tests** en générant du trafic (par exemple, avec ping et nmap) pour valider que la capture, l'enregistrement en base de données et l'affichage sur l'interface fonctionnaient correctement. J'ai également dû surmonter des difficultés techniques, notamment liées aux droits d'accès nécessaires pour la capture de paquets et à la gestion des performances de la base de données face à un volume de trafic important.

# Contexte implémentation :

Pour les **entreprises**, l'utilisation d'un sniffer réseau s'avère précieuse pour plusieurs raisons :

* **Tester l'infrastructure réseau :** En observant le flux de données, les administrateurs peuvent vérifier si le réseau fonctionne comme prévu, identifier les éventuels goulots d'étranglement, les latences excessives ou les erreurs de transmission qui pourraient impacter la performance globale.
* **Identifier les problèmes de réseau :** Un sniffer aide à diagnostiquer les problèmes de connectivité, les dysfonctionnements d'applications ou les erreurs de configuration en analysant les échanges de paquets et en repérant les anomalies ou les échecs de communication.
* **Trouver des brèches de sécurité :** L'analyse du trafic peut révéler des activités suspectes, telles que des tentatives d'accès non autorisées, des transferts de données sensibles en clair (non chiffrées), la présence de logiciels malveillants communiquant avec l'extérieur, ou des scans de ports. Cela permet d'identifier les vulnérabilités exploitables par des acteurs malveillants.

Pour les **particuliers**, bien que l'échelle soit différente, les bénéfices sont similaires :

* **Comprendre l'activité de son réseau domestique :** Un particulier curieux peut utiliser un sniffer pour visualiser les appareils connectés et le type de trafic qu'ils génèrent.
* **Résoudre des problèmes de connexion :** En cas de lenteur ou de coupures, l'analyse du trafic peut aider à identifier la source du problème.
* **Détecter des appareils inconnus ou des activités suspectes :** Un sniffer peut alerter sur la présence d'appareils non autorisés sur le réseau ou sur des communications inattendues émanant de certains périphériques, potentiellement signe d'une compromission.

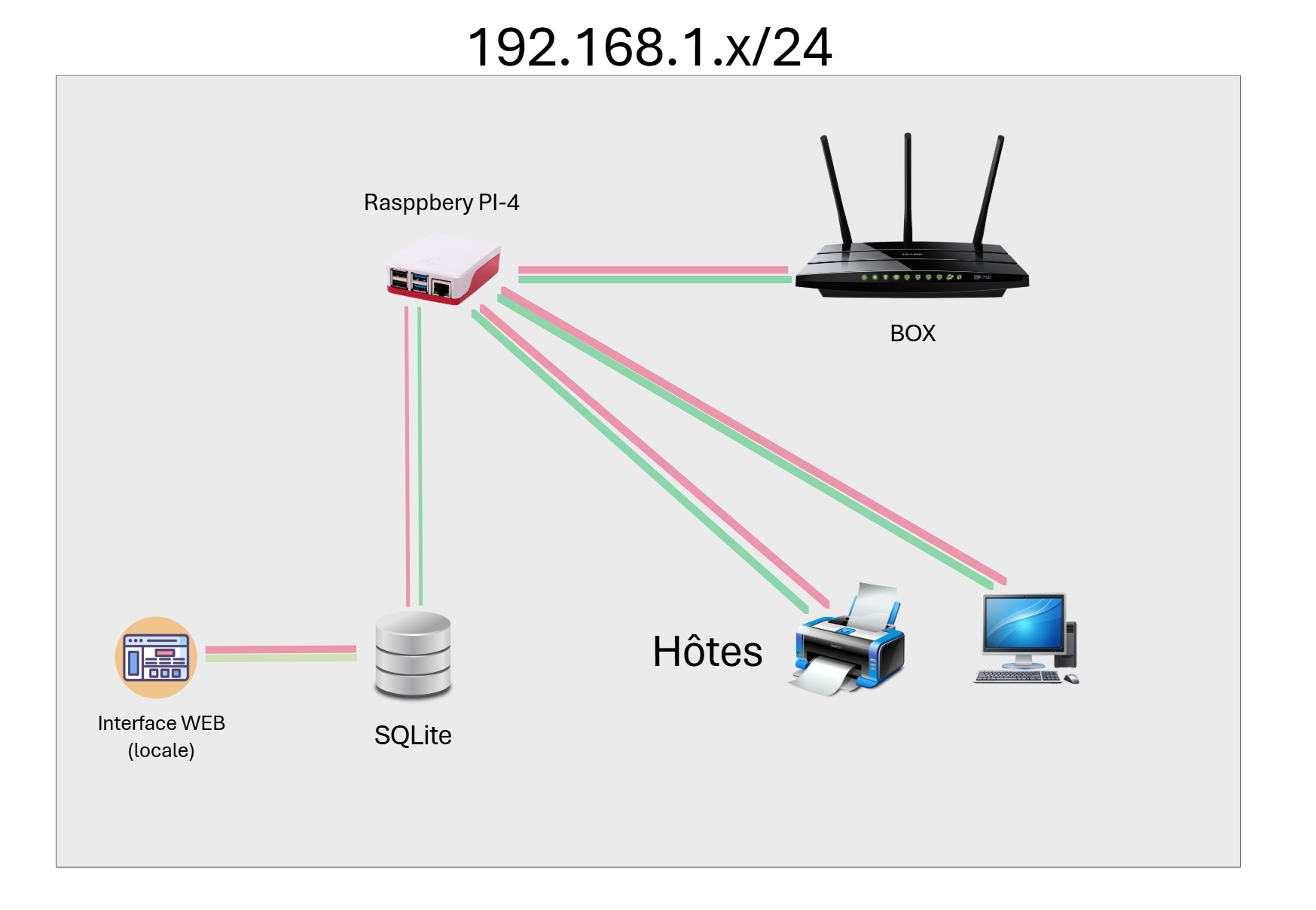
*(L'utilisation du sniffer réseau doit se faire de manière éthique et légale, en respectant la vie privée des utilisateurs et en ayant les autorisations nécessaires pour capturer et analyser le trafic sur un réseau donné.)*

# Cahier des charges :

L'objectif principal est de développer et déployer une solution de sniffer réseau sur un Raspberry Pi 4 pour permettre la surveillance en temps réel du trafic d'un petit réseau local. Cette solution doit être légère, économique et offrir une interface intuitive pour l'analyse des données capturées, tout en garantissant la sécurité des accès.

|  |  |
| --- | --- |
| **Capture de Trafic Temps Réel** | La solution doit être capable de capturer les trames réseau passant par l'interface du Raspberry Pi 4 en temps réel. |
| **Stockage des Données** | Les informations extraites des trames capturées (horodatage, adresses MAC/IP source et destination, ports source et destination, protocole, taille du paquet, éventuellement des informations de couche application si pertinent et configuré) sont stockées de manière structurée dans une base de données. |
| **Interface Web** | Une interface utilisateur accessible via un navigateur web doit permettre de visualiser les données capturées de manière claire et organisée. |
| **Recherche et Filtrage dans l'Interface Web** | L'interface web doit proposer des fonctionnalités de recherche et de filtrage pour faciliter l'analyse des données stockées. |
| **Sécurité de l'Interface Web** | L'accès à l'interface web doit être sécurisé, a minima par une authentification (nom d'utilisateur/mot de passe). |

# Topologie :



# Explications concert :

## 4.1. Mise en place de la base de données

Une base de données SQLite a été choisie pour stocker de manière persistante les informations pertinentes extraites des trames réseau capturées. Un script Python a été utilisé pour créer la table packets si elle n'existe pas déjà. Cette table est conçue pour enregistrer l'horodatage, les adresses IP source et destination, le protocole et la taille de chaque paquet intercepté.

Voici l'exemple de code utilisé pour la création de la table :

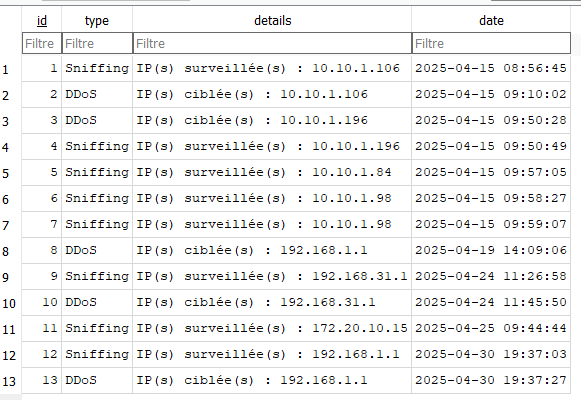
Python

import sqlite3

conn = sqlite3.connect('network.db')

c = conn.cursor()

c.execute('''  
CREATE TABLE IF NOT EXISTS packets (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 timestamp TEXT,  
 src\_ip TEXT,  
 dst\_ip TEXT,  
 protocol TEXT,  
 length INTEGER  
)  
''')  
conn.commit()

conn.close()

Database:

## 4.2. Développement de l’interface web

Une interface web est développée à l'aide du framework Python Flask pour permettre la visualisation et l'administration des données de trafic capturées. Une route initiale (/) a été définie pour afficher les 100 dernières trames enregistrées dans la base de données.

Voici l'exemple de code de la route principale de l'application Flask :

from flask import Flask, render\_template

import sqlite3

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

conn = sqlite3.connect('network.db')

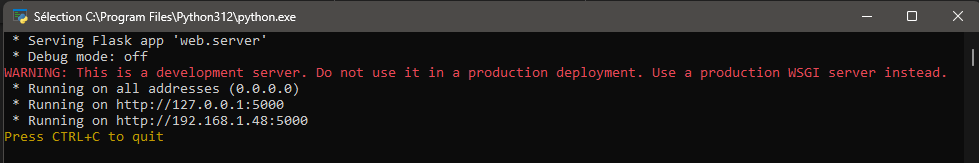
c = conn.cursor()

c.execute('SELECT \* FROM packets ORDER BY id DESC LIMIT 100')

packets = c.fetchall()

conn.close()

return render\_template('index.html', packets=packets)

**

## 4.3. Capture et analyse des trames réseau

La capture et l'analyse en temps réel des trames réseau sont réalisées en utilisant la librairie Python Scapy. Une fonction process\_packet est définie pour traiter chaque paquet intercepté. Si le paquet contient une couche IP, les informations pertinentes (horodatage, IPs source et destination, protocole, taille) sont extraites et insérées dans la base de données SQLite. La fonction sniff de Scapy est utilisée pour démarrer la capture en continu, en appelant process\_packet pour chaque trame.

Voici l'exemple de code de capture et de traitement des trames :

from scapy.all import sniff, IP

import sqlite3

from datetime import datetime

def process\_packet(packet):

if IP in packet:

conn = sqlite3.connect('network.db')

c = conn.cursor()

c.execute('''

INSERT INTO packets (timestamp, src\_ip, dst\_ip, protocol, length)

VALUES (?, ?, ?, ?, ?)

''', (

datetime.now().strftime('%Y-%m-%d %H:%M:%S'),

packet[IP].src,

packet[IP].dst,

packet[IP].proto,

len(packet)

))

conn.commit()

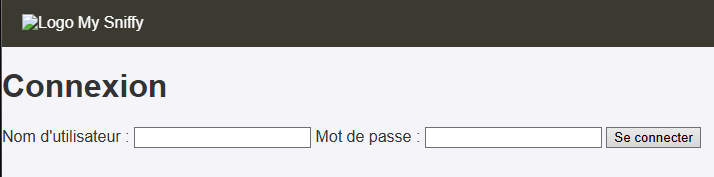
conn.close()

sniff(prn=process\_packet, store=0)

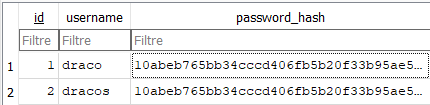
# Sécurité et gestion des accès

L’accès à l’interface web est protégé par un système d’authentification simple (login/mot de passe). Qui se trouve dans la base de donner User en étant hash avec ‘’sah256’’

Point de vue web :



Vue Database :



# Difficultés Rencontrées et Solutions

La mise en œuvre de ce projet a présenté certains défis techniques, typiques du développement sur des plateformes embarquées avec des contraintes de ressources et de sécurité. Voici les principales difficultés rencontrées et les solutions adoptées :

## Problème de Droits Root pour la Capture de Paquets :

**Difficulté :** La capture de paquets réseau bruts nécessite généralement des privilèges élevés (root) sur un système Linux pour accéder directement à l'interface réseau en mode promiscuous. Exécuter des scripts Python avec Scapy sans ces droits a causé des erreurs de permission.

**Solution :** L'exécution du script de capture de paquets avec sudo a été la solution immédiate pour accorder les droits nécessaires à Scapy. *(Il est à noter que pour une solution de production, des méthodes plus sécurisées comme l'attribution de capacités spécifiques à l'exécutable Python ou l'exécution du sniffer par un utilisateur dédié avec des droits restreints pourraient être envisagées pour minimiser les risques liés à l'exécution en tant que root en continu).*

## Performance de la Base de Données avec un Grand Volume de Paquets :

**Difficulté :** Avec la capture continue de trafic, la base de données SQLite peut rapidement se remplir. L'affichage d'un très grand nombre d'enregistrements sur l'interface web peut entraîner des ralentissements significatifs et une consommation importante de mémoire sur le Raspberry Pi.

**Solution :** Pour atténuer ce problème de performance lors de la consultation, l'interface web a été limitée à l'affichage des 100 derniers paquets capturés (comme vu dans l'exemple de code Flask). Cette approche permet de visualiser l'activité réseau récente sans surcharger le système. *(Pour gérer le volume de données sur le long terme, des stratégies d'archivage, de rotation de base de données ou de suppression des enregistrements les plus anciens seraient nécessaires dans une version plus évoluée).*

## Sécurisation de l’Interface Web :

**Difficulté :** L'exposition d'une interface d'administration web, même en réseau local, représente un point d'entrée potentiel pour des accès non autorisés si elle n'est pas protégée.

**Solution :** Un système d'authentification simple basé sur un login et un mot de passe a été mis en place pour restreindre l'accès à l'interface web. Ceci constitue une première barrière de sécurité essentielle.

## Difficultés liées à la Mise en Place de Flask, HTML, CSS, JS :

**Difficulté :** L'intégration et le développement de l'application web en utilisant le framework Flask pour le backend et les technologies web classiques (HTML, CSS, JavaScript) pour le frontend ont également représenté des défis. Cela inclut potentiellement la gestion des requêtes HTTP, le rendu des templates HTML, l'interaction entre le frontend et le backend via JavaScript, et le stylisme de l'interface avec CSS.

**Solution :** Ces difficultés ont été surmontées grâce au travail de développement, à la documentation disponible pour ces technologies et, potentiellement, à des phases de débogage et d'itération pour assurer la bonne communication entre les différentes couches de l'application web.

# perspectives amelioration

Le projet actuel constitue une base solide pour la surveillance de réseau local. Plusieurs pistes d'amélioration peuvent être explorées pour enrichir ses fonctionnalités, renforcer sa robustesse et accroître sa valeur, potentiellement en vue d'évoluer vers un système de détection d'intrusion plus sophistiqué :

* **Ajout d’un Système d’Alerte en Cas de Détection de Trafic Suspect :**
  + **Description :** Développer des règles ou des heuristiques pour identifier des modèles de trafic anormaux ou potentiellement malveillants (par exemple, un grand nombre de connexions échouées vers un port spécifique, un volume de données inhabituel, des communications avec des adresses IP connues pour être malveillantes).
  + **Bénéfice :** Permettrait de notifier proactivement l'administrateur en cas d'incident de sécurité potentiel, réduisant le temps de réaction face aux menaces.
* **Passage à une Base de Données plus Robuste (MySQL/PostgreSQL) :**
  + **Description :** Migrer le stockage des données de SQLite vers un système de gestion de base de données client-serveur plus performant et scalable comme MySQL ou PostgreSQL.
  + **Bénéfice :** Améliorerait significativement la performance et la fiabilité pour gérer de plus grands volumes de données de trafic sur le long terme, et faciliterait potentiellement l'accès aux données depuis d'autres applications ou services.
* **Déploiement d’un Certificat SSL/TLS pour Sécuriser l’Interface Web :**
  + **Description :** Configurer le serveur web (Nginx/Apache) sur le Raspberry Pi pour utiliser HTTPS en installant un certificat SSL/TLS.
  + **Bénéfice :** Assurerait le chiffrement de toutes les communications entre le navigateur de l'utilisateur et l'interface web, protégeant les identifiants et les données consultées contre les écoutes clandestines, renforçant ainsi la sécurité d'accès.
* **Analyse plus Poussée des Trames (ex : Détection d’Attaques) :**
  + **Description :** Aller au-delà de la simple extraction d'informations de base en analysant plus en profondeur le contenu et les caractéristiques des paquets (par exemple, analyser les en-têtes de protocoles spécifiques comme HTTP, DNS, ou analyser les payloads pour détecter des signatures d'attaques connues).
  + **Bénéfice :** Permettrait d'identifier des menaces plus sophistiquées, telles que des tentatives d'injection SQL, des débordements de tampon, ou des communications associées à des malwares.

# Conclusion

En synthèse, ce projet a démontré la faisabilité et l'efficacité de la mise en place d'une solution de surveillance réseau légère et économique en utilisant un Raspberry Pi 4 comme plateforme matérielle centrale. L'intégration d'outils open source comme Scapy pour la capture de paquets, SQLite pour le stockage des données et Flask pour l'interface web a permis de construire un système fonctionnel capable de monitorer le trafic d'un petit réseau local en temps réel.

Cette solution offre aux entreprises et aux particuliers un moyen accessible de mieux comprendre l'activité de leur réseau, d'identifier les problèmes potentiels et de commencer à détecter des signes avant-coureurs de comportement anormal ou potentiellement malveillant.

Au-delà de la surveillance basique, cette architecture fournit une base solide et flexible qui ouvre la voie à des évolutions futures significatives. Il serait possible d'enrichir l'analyse des paquets pour détecter des schémas d'attaque spécifiques, d'intégrer des règles de corrélation d'événements, ou encore d'ajouter des fonctionnalités d'alerte. Ainsi, ce projet de sniffer réseau peut évoluer progressivement vers un système de détection d'intrusion (IDS) plus avancé, renforçant d'autant la sécurité de l'infrastructure surveillée.

# Annexes

* **Code source** : voir fichiers db\_init.py, sniffer.py, app.py.
* **Captures d’écran** : interface web, scripts Python.
* **Bibliographie** : documentation officielle de Scapy, Flask, SQLite, tutoriels Raspberry Pi.