Simulation et Génération de Traces d'Un Réseaux Domestique

Nom : KUATE KAMGA BRAYAN A.
Département du Génie Informatique, Polytechnique Yaoundé

30 novembre 2025

Résumé

Ce rapport présente la méthodologie et les résultats de la simulation d'un réseau domestique connecté réalisée sous `ns-3`. L'objectif est de générer un jeu de données (dataset) qui semble le plus réaliste possible, permettant l'entraînement de modèles de Machine Learning pour la classification de trafic. Nous avons detaillé ici les choix de modélisation, l'introduction de variabilité spatiale et temporelle, ainsi que la chaîne de traitement des données.

1 Introduction

Dans le contexte croissant de l'Internet des Objets (IoT), la sécurité et la gestion de la qualité de service (QoS) nécessitent une identification précise des applications circulant sur le réseau. Ce projet vise à simuler un environnement domestique dense et réaliste afin de constituer un jeu de données (*dataset*) destiné à l'entraînement d'algorithmes de classification par Machine Learning (ML). Contrairement à des traces statiques, notre approche privilégie la variabilité des paramètres réseaux et applicatifs pour garantir la robustesse du modèle futur.

2 Modélisation Scientifique et Scénarios

Pour représenter fidèlement un réseau domestique, nous avons modélisé $N = [10, 30]$ équipements répartis en $K = 4$ classes d'applications distinctes. Chaque classe possède une signature trafic spécifique en termes de protocole de transport, de débit et de modèle temporel.

2.1 Typologie des Applications

Le tableau 1 résume les caractéristiques techniques implémentées pour chaque type de trafic.

Type (K)	**Application**	**Comportement (Pattern)**	**Protocole**
Type 0	Capteur IoT	Sporadique, très petits paquets (40-80B), débit minimal.	UDP
Type 1	Caméra IP	Flux continu (CBR), gros paquets, débit élevé (HD).	UDP
Type 2	Navigateur Web	Trafic en rafales (Burst), modèle ON/OFF (LogNormal), réponse TCP.	TCP
Type 3	Appel VoIP	Flux temps-réel, paquets moyens, intervalles réguliers.	UDP

Table 1 – Caractéristiques des profils de trafic simulés.

2.2 Stratégie d'Introduction de la Variabilité

Un défi majeur des simulations est leur déterminisme, qui nuit à l'apprentissage des modèles ML (risque d'overfitting). Pour répondre à l'exigence de variabilité, nous avons introduit de l'aléatoire à trois niveaux :

2.2.1 Variabilité Spatiale (Hardware)

Chaque équipement ne se comporte pas exactement comme son voisin malgré qu'ils soient du même type. Lors de l'instanciation, nous tirons aléatoirement la taille des paquets (ex : entre 1000 et 1400 octets pour les caméras) et les débits cibles au sein d'une plage définie. L

TABLE 2 – Paramètres de variabilité configurés par type d'application

Type	Application	Protocole	Plage Taille Paquets	Débit
Type 0	Capteur IoT	UDP	40 – 80 Octets	$\approx$ 1 kbps (Sporadique)
Type 1	Caméra IP	UDP	1000 – 1400 Octets	1.5 – 2.5 Mbps (Continu)
Type 2	Navigateur Web	TCP	500 – 1460 Octets	5 Mbps (Rafales)
Type 3	Appel VoIP	UDP	160 – 200 Octets	64 kbps (Régulier)

2.2.2 Variabilité Temporelle (Comportement) :

L'instant de démarrage de chaque nœud est aléatoire uniformément sur $[0, 5s]$. Les temps de "silence" (OFF state) suivent des lois exponentielles pour simuler l'humain ou l'environnement.

2.2.3 Variabilité Physique (Environnement) :

Au niveau de la couche PHY 802.11ac, nous utilisons un modèle de perte de propagation **Nakagami**. Contrairement à un modèle déterministe, il simule des fluctuations rapides du signal (Fast Fading), forçant le Wifi à adapter ses schémas de modulation (MCS) dynamiquement, simulant ainsi des traces qui tendent vers la réalité.

3 Architecture de Simulation sous ns-3

La simulation est réalisée avec `ns-3` (version 3.45). L'architecture réseau, illustrée par la topologie NetAnim, se compose :

- D'un **Point d'Accès (AP)** central configuré en 802.11ac avec l'algorithme d'adaptation de lien `MinstrelHt`.
- De $N = [10 - 30]$ **stations Wifi** placées selon une disposition d'alignement vertical comme illustré dans le document qui presente le travail à faire. Lors de la dernière étape de notre chaine de traitement, nous avons opté pour une disposition aléatoire pour varier les distances des stations Wifi vis à vis du point d'accès.
- De $K = 4$ **serveurs distants** connectés à l'AP via un lien Ethernet Gigabit (CSMA). Chaque type d'application communique exclusivement avec son serveur dédié, ce qui permet un étiquetage (Labeling) automatique basé sur l'IP de destination.

Nous activons le module `PcapTracing` sur l'interface Wifi de l'AP pour capturer l'intégralité du trafic radio, incluant les en-têtes de gestion (Beacons, Ack) et les données.

4 Ingénierie des Données (Data Engineering)

Les traces brutes (`.pcap`) contiennent trop de bruit pour être utilisées telles quelles. Nous avons développé un pipeline de traitement en Python (utilisant `Scapy` et `Pandas`) pour transformer ces traces en un dataset structuré.

4.1 Extraction de Caractéristiques (Feature Engineering)

Plutôt que d'analyser chaque paquet isolément, nous agrégeons le trafic par flux et par fenêtres temporelles (ex : 1 seconde). Pour chaque fenêtre, nous calculons des métriques statistiques avancées :

- **IAT (Inter-Arrival Time) :** Moyenne et écart-type des temps entre deux paquets consécutifs. C'est la métrique discriminante pour distinguer un automate (IoT régulier) d'un humain (Web irrégulier).
- **Volumétrie :** Taille moyenne, min, max et débit total.
- **Protocole :** Drapeaux TCP/UDP.

L'étiquetage (*Labeling*) est réalisé automatiquement grâce à la connaissance de la topologie : si un paquet est destiné au serveur 192.168.1.3, il est étiqueté "Camera_IP".

5 Résultats et Validation de la Chaîne de Traitement

Notre approche s'est voulue incrémentale. Nous avons d'abord validé la topologie et la capture de données brutes avant d'enrichir le modèle avec de la variabilité et un post-traitement avancé.

5.1 Validation Visuelle et Topologique (NetAnim)

Nous avons validé l'infrastructure de notre réseau de facon incrémentale, permettant de vérifier visuellement le comportement du réseau avant de lancer des simulations massives. L'approche incrementale a été adopté pour des raisons de prise en main de l'outil **ns-3**. Cette étape s'est appuyée sur l'outil `NetAnim`.

Étape 1 : Topologie de Base (`wifi-data-generation1.cc`) Dans un premier temps, nous avons implémenté une architecture simplifiée comprenant $N = 10$ équipements identiques connectés à un unique serveur distant. L'animation générée (`maison_animation1.xml`) a permis de valider le bon fonctionnement du routage WiFi vers Ethernet.

Étape 2 : Architecture Multi-Services (`wifi-data-generation2.cc`) Nous avons ensuite complexifié le scénario en introduisant les $K = 4$ types d'applications et les K serveurs distincts. L'analyse du fichier `maison_animation2.xml` nous a permis de confirmer deux points critiques :

- **Le Layout :** Les stations sont alignées verticalement à gauche (Zone domestique), l'AP est au centre, et les serveurs sont alignés à droite (Zone Internet/Cloud).
- **Le Routage Applicatif :** Les flèches de transmission (vecteurs verts) confirment que chaque type d'équipement adresse bien son serveur dédié (ex : un nœud "Caméra" envoie ses paquets vers le Serveur 1, un nœud "IoT" vers le Serveur 0).

Note : Pour les simulations finales à haute densité ($N = 30$) et longue durée ($simulationTime = 60$), les animations des simulations n'ont pas pu être visualisées, la taille des fichiers XML résultants étant trop volumineuse pour être traitée efficacement par NetAnim.

5.2 Résultats Intermédiaires : Capture et Dataset "Paquet par Paquet"

Une fois la topologie validée, l'objectif de la troisième itération (`wifi-data-generation3.cc`) était d'activer la collecte de données.

5.2.1 Mise en place de la capture

Nous avons introduit la fonctionnalité `PcapTracing` au niveau de la couche physique du Point d'Accès. Ce choix stratégique permet de capturer l'intégralité du trafic (entrant et sortant) transitant par le cœur du réseau domestique. La simulation, exécutée sur $N = 30$ nœuds pendant 60 secondes, a produit le fichier de traces brutes `wifi-traces-30-1.pcap`.

5.2.2 Structure du Dataset Intermédiaire

Ces traces ont été traitées par notre premier script d'extraction `pcap_to_dataset.py`. Ce script convertit chaque paquet capturé en une ligne de données CSV (`dataset1.csv`). Le tableau ci-dessous détaille les attributs extraits pour chaque paquet :

- **Timestamp :** L'instant précis de la capture (en secondes).
- **Source_IP / Dest_IP :** Les adresses identifiant l'émetteur et le récepteur. C'est ici que se fait l'étiquetage primaire (ex : si Dest_IP est le Serveur Caméra, le paquet est étiqueté "Camera").
- **Source_Port / Dest_Port :** Les ports de transport (discriminant pour certains services).
- **Protocol :** Le protocole de transport détecté (TCP ou UDP).
- **Size_Bytes :** La taille totale du paquet capturé.
- **Label :** La classe de vérité terrain (Ground Truth) déduite de l'architecture.

5.2.3 Limites de cette approche

Bien que ce dataset intermédiaire valide la chaîne de collecte, il présente une limite majeure pour le Machine Learning : l'absence de contexte temporel. Un paquet UDP isolé de 1000 octets ne contient pas assez d'information pour être classifié avec certitude. C'est pourquoi nous avons développé une version finale introduisant le fenêtrage.

5.3 Résultats Finaux : Dataset "Statistique" avec Variabilité

Pour pallier l'absence de contexte temporel du premier dataset et répondre à l'exigence de classification sur des « morceaux de trafic », nous avons implémenté une approche par agrégation statistique. Cette étape repose sur la simulation (`wifi-data-generation4.cc`) et un script python de traitement repensé.

5.3.1 Introduction de la Variabilité Réaliste

Avant de générer le dataset final, il était impératif de briser le déterminisme de la simulation. Dans `wifi-data-generation4`, nous avons introduit :

- **Variabilité Physique :** Ajout du modèle de perte `NakagamiPropagationLoss`. Il simule des fluctuations rapides du signal (Fast Fading), forçant le Wifi à adapter ses débits.
- **Variabilité Applicative :** Les tailles de paquets et les débits ne sont plus fixes mais tirés aléatoirement pour chaque nœud dans des plages définies (ex : une caméra envoie des paquets de 1020 octets, une autre de 1350 octets).

La simulation finale, exécutée sur $N = 25$ nœuds pendant 60 secondes, a produit le fichier de traces brutes **wifi-traces-25-1.pcap**.

5.3.2 Transformation : Du Paquet à la Fenêtre Temporelle

Le script `pcap_to_dataset_final.py` implémente une logique de **fenêtrage glissant** (*Sliding Window*) d'une durée paramétrable de $\Delta t = 1.0$ seconde. Au lieu de traiter chaque paquet individuellement, le script regroupe les paquets appartenant au même flux (couple IP Source / IP Destination) à l'intérieur de chaque fenêtre temporelle.

5.3.3 Structure du Dataset Final

Le fichier résultant, `dataset_final.csv`, constitue le livrable principal. Chaque ligne représente l'activité d'une application pendant une seconde. Les attributs (Features) ont été choisis pour capturer la signature comportementale du trafic :

1. **Identification :**
   — `flow_id` : Identifiant unique du flux.
   — `window_id` : Indice temporel de la fenêtre (séquence).
   — `proto` : Protocole de transport (6 pour TCP/ 17 pour UDP).
2. **Volumétrie (Charge) :**
   — `pkt_count` : Nombre de paquets reçus par seconde. Permet de distinguer les applications bavardes (Caméra) des silencieuses (IoT).
   — `bytes_total` : Bande passante consommée sur la fenêtre.
   — `size_mean` / `size_std` : Moyenne et écart-type de la taille des paquets.
3. **Temporalité (Comportement) :**
   — `iat_mean` : Temps moyen entre deux paquets consécutifs (Inter-Arrival Time).
   — `iat_std` : Écart-type des temps d'arrivée. C'est la métrique la plus discriminante.
      — Une valeur proche de 0 indique que le trafic est très régulier. C'est une machine (VoIP, Caméra)
      — Une valeur élevée indique que Le trafic est irrégulier. C'est un comportement humain sur le Web.
4. **Cible (Target) :**
   — `Label` : La classe à prédire (`IoT_Sensor`, `Camera_IP`, `Web_Traffic`, `VoIP_Call`).

6 Reproductibilité

L'ensemble du projet respecte la structure standard de l'environnement **ns-3**. Voici les instructions pour localiser les fichiers et reproduire les expériences.

6.1 Guide de Reproduction

Les différents scripts C++ illustrant l'évolution du projet (de la topologie simple à la version finale) sont situés dans le répertoire **scratch** du simulateur :

```
~/workspace/ns-allinone-3.45/ns-3-dev/scratch
```

Pour lancer une simulation (par exemple la version finale), exécutez la commande suivante depuis la racine du dossier **ns-3-dev** :

```
./ns3 run "scratch/wifi-data-generation4 --nWifi=25"
```

Les paramètres optionnels `-verbose=true` ou `-simulationTime=60` peuvent être ajoutés pour ajuster l'exécution. Une fois la simulation terminée, *les fichiers résultats (.pcap, .xml)* sont générés directement à la racine du dossier **ns-3-dev**

Le dataset final (dataset_final.csv) se trouve dans le repertoire "workspace/analysis". Il a été généré à partir du fichier *wifi-traces-25-1.pcap*

6.2 Dépôt GitHub

L'intégralité du code source (C++, Python), ainsi que les instructions de compilation et les fichiers de configuration, sont disponibles sur le dépôt suivant :

`https://github.com/KamgaBrayan/domestic-network-simulation-ns3.git`

Ce dépôt contient également un échantillon des traces générées (.pcap) et le dataset final (.csv).