Analyse avancée des données NMS grâce au Machine Learning pour la prédiction d'anomalies

[PRFE'25] Learning NMS

Abderaouf BIREM
Maxime GANNE
Henri THOMAS

Avec la supervision de M. Nidà Meddouri, Docteur en Machine Learning et Chercheur au LRE*

Avril 2025

Contents

*Laboratoire de Recherche de l'EPITA

1 Contexte et Problématique

1.1 Limites des approches traditionnelles de supervision réseau

Les systèmes de gestion de réseau (*Network Management Systems*, NMS) jouent un rôle central dans l'administration des infrastructures informatiques. Ils assurent la supervision des équipements (routeurs, commutateurs, serveurs, etc.) en collectant des données de télémétrie, par exemple via les protocoles SNMP pour les métriques d'équipement ou NetFlow pour les flux de trafic, et en générant des alarmes en cas d'anomalie. L'objectif est de détecter rapidement les pannes ou dégradations de performance afin de garantir la disponibilité et la qualité de service du réseau.

Cependant, les méthodes traditionnelles de supervision montrent leurs limites face à l'évolution des réseaux modernes. La complexité croissante (virtualisation, cloud, IoT, etc.) s'accompagne d'un volume massif de données à analyser, rendant difficile pour des opérateurs humains d'identifier proactivement les incidents critiques. Les NMS conventionnels reposent sur des règles statiques et des seuils prédéfinis pour déclencher des alertes, une approche qui peine à anticiper les problèmes et génère souvent des faux positifs.[10] Par exemple, un pic de trafic momentanément au-dessus d'un seuil peut déclencher une alarme sans que cela corresponde à un réel incident, tandis qu'à l'inverse, une dégradation progressive peut passer inaperçue si aucun seuil n'est adéquat.

Plusieurs limitations apparaissent donc avec ces solutions traditionnelles :

- une approche essentiellement réactive, sans réelle capacité prédictive ;
- une surcharge d'alarmes rendant difficile la distinction des événements importants ;
- une incapacité à détecter des motifs complexes ou inconnus d'anomalies dans le flot de données hétérogènes du réseau.

1.2 Vers une supervision intelligente par apprentissage automatique

Dans ce contexte, il devient nécessaire d'envisager des solutions plus intelligentes et automatiques. L'intelligence artificielle et le *machine learning* (ML) offrent des pistes prometteuses pour relever ces défis en apportant des capacités d'analyse avancée. En effet, des algorithmes de ML peuvent apprendre à partir de larges volumes de données et détecter des schémas inhabituels ou des signes avant-coureurs d'un incident futur.[10] Contrairement aux règles fixes, un modèle d'apprentissage peut mettre en évidence des corrélations subtiles entre métriques ou repérer une anomalie par rapport au comportement historique normal du réseau. Ainsi, l'intégration du ML au sein des NMS pourrait permettre de prévoir des pannes ou congestions avant qu'elles n'affectent les utilisateurs, d'automatiser la détection d'événements critiques en temps réel, et d'améliorer la pertinence des alertes en réduisant le bruit.

C'est dans cette optique qu'est posée la problématique du projet *Learning NMS* : **comment exploiter le machine learning pour améliorer la détection et la prédiction d'événements réseau critiques à partir des données de télémétrie collectées par un NMS ?**

2 Le Machine Learning appliqué aux NMS

2.1 Méthodes classiques et avancées de détection d'anomalies

De nombreux travaux de recherche récents se sont attachés à appliquer le *Machine Learning* à la gestion de réseau, en particulier pour la détection d'anomalies et la prédiction de pannes à partir des données de télémétrie. Parmi les méthodes classiques explorées figurent les réseaux bayésiens, les arbres de décision, les machines à vecteurs de support (SVM) ou encore les forêts d'arbres aléatoires. Ces techniques permettent de construire des modèles à partir de données historiques du réseau et d'identifier ensuite tout écart significatif par rapport au comportement attendu.

Par exemple, une approche bayésienne a été proposée pour doter un NMS de capacités autonomes de décision : en formulant un modèle de réseau bayésien apprenant, un système de management peut prédire

les conditions de défaillance et prendre des décisions proactives. Dans le cas de l'admission de nouveaux flux dans un cœur de réseau télécom, un NMS piloté par réseau bayésien a montré sa capacité à anticiper et à améliorer la gestion des appels par rapport aux politiques statiques traditionnelles.[7]

D'autres travaux se sont focalisés sur l'analyse statistique des données SNMP collectées par les NMS : en appliquant des classifieurs supervisés aux compteurs d'équipement (MIB), il est possible de détecter des écarts correspondant à des défaillances ou des attaques. Globalement, ces approches de ML se sont révélées efficaces pour distinguer les comportements normaux de ceux anormaux dans le réseau, obtenant des taux de détection élevés lors des expérimentations.[5] Par exemple, Al-Kasassbeh *et al.* ont utilisé trois algorithmes (un arbre de décision C4.5/J48, un *Random Forest* et un arbre REP) pour classer des instances issues de données SNMP simulant des attaques par déni de service ; ils rapportent que le classifieur REP Tree a obtenu la meilleure précision de détection d'anomalies sur l'ensemble de test.[9] De même, Fosić *et al.* ont évalué plusieurs algorithmes de classification (SVM, k-plus proches voisins, *Naïve Bayes*, arbre de décision, forêt aléatoire, AdaBoost, etc.) sur des flux réseau (dataset public UNSW-NB15) représentant du trafic normal et malveillant : dans leur étude, la forêt aléatoire atteint la meilleure performance (F2-score $\approx$ 97,7% et AUC $\approx$ 98,5%), illustrant la capacité de ces modèles à détecter avec précision les anomalies tout en tenant compte du fort déséquilibre des données.[8] Les SVM ont également été investigués pour la détection d'intrusions réseau, y compris sous forme de *one-class SVM* pour repérer des comportements hors du profil normal attendu, mais les résultats montrent souvent que des modèles ensemblistes ou arborescents peuvent mieux s'adapter aux données de télémétrie et fournir une détection plus robuste dans ce contexte.[9][8]

En parallèle des techniques classiques, l'essor du *deep learning* a ouvert de nouvelles perspectives pour les NMS. Des réseaux de neurones peuvent en effet capturer des relations complexes non linéaires entre métriques et potentiellement améliorer la détection d'anomalies subtiles. Par exemple, une étude récente a proposé l'utilisation d'un auto-encodeur empilé (*Stacked Autoencoder*) entraîné sur des données SNMP afin d'identifier des anomalies réseau.[6] Ce modèle non supervisé apprend à reconstruire les profils de trafic normaux et signale toute divergence importante. Appliqué à un jeu de données SNMP-MIB contenant des attaques simulées de type déni de service (DoS), l'auto-encodeur a atteint un taux de détection de 100% des anomalies connues, surpassant ainsi 22 autres techniques de *machine learning* testées sur le même jeu de données.[6]

2.2 Limites actuelles et originalité de l'approche proposée

Malgré ces avancées, il subsiste des défis pour intégrer le ML dans les NMS opérationnels. D'une part, bon nombre de travaux de recherche ciblent des cas d'usage spécifiques (souvent la détection d'intrusions ou d'attaques réseau) et ne traitent qu'une partie de la télémétrie disponible. Par exemple, les études d'*anomaly detection* se concentrent fréquemment soit sur les données de flux (NetFlow/IPFIX), soit sur les compteurs SNMP, mais rarement sur les deux de façon conjointe. D'autre part, la plupart des approches nécessitent des données étiquetées pour entraîner les modèles supervisés, or il est difficile de constituer des jeux de données représentatifs d'incidents réseau réels (les pannes critiques étant heureusement peu fréquentes et souvent confidentielles). Cette difficulté a conduit certains chercheurs à générer artificiellement des datasets pour leurs expérimentations – par exemple en simulant des attaques afin d'alimenter un corpus SNMP MIB factice.[9]

L'originalité de notre projet *Learning NMS* réside précisément dans la mise en place d'une infrastructure simulée permettant de combiner plusieurs sources de données NMS et de créer un dataset annoté couvrant divers scénarios d'anomalies. Contrairement aux études limitées à un seul type de données, nous exploitons à la fois les métriques SNMP (issues de requêtes *poll* et de *traps*) et les informations de flux NetFlow pour entraîner le modèle de ML. Cette approche holistique fournit une vision plus complète de l'état du réseau à chaque instant. De plus, en injectant des anomalies connues dans la simulation (telles que des pannes d'interface, des surtensions de trafic, des attaques simulées, etc.), nous disposons d'une véritable *ground truth* pour évaluer les algorithmes de détection. La solution proposée apparaît ainsi à la fois originale et pertinente : elle s'appuie sur des données de télémétrie réalistes (proches de celles d'un NMS en production) tout en permettant l'évaluation contrôlée de méthodes d'apprentissage automatique dans un environnement de test reproductible.

3 Étude expérimentale

Afin de valider notre approche, nous avons conçu une infrastructure réseau simulée dans GNS3 reproduisant un scénario simplifié mais réaliste de réseau d'entreprise.

3.1 Infrastructure et configuration des équipements

Cette infrastructure se compose de deux machines virtuelles Ubuntu 22.04 Server (2 Go RAM, 1 vCPU) déployées sur VirtualBox, interconnectées par un routeur Cisco 3600 virtualisé. L'ensemble est supervisé par une troisième machine virtuelle jouant le rôle de station de supervision.

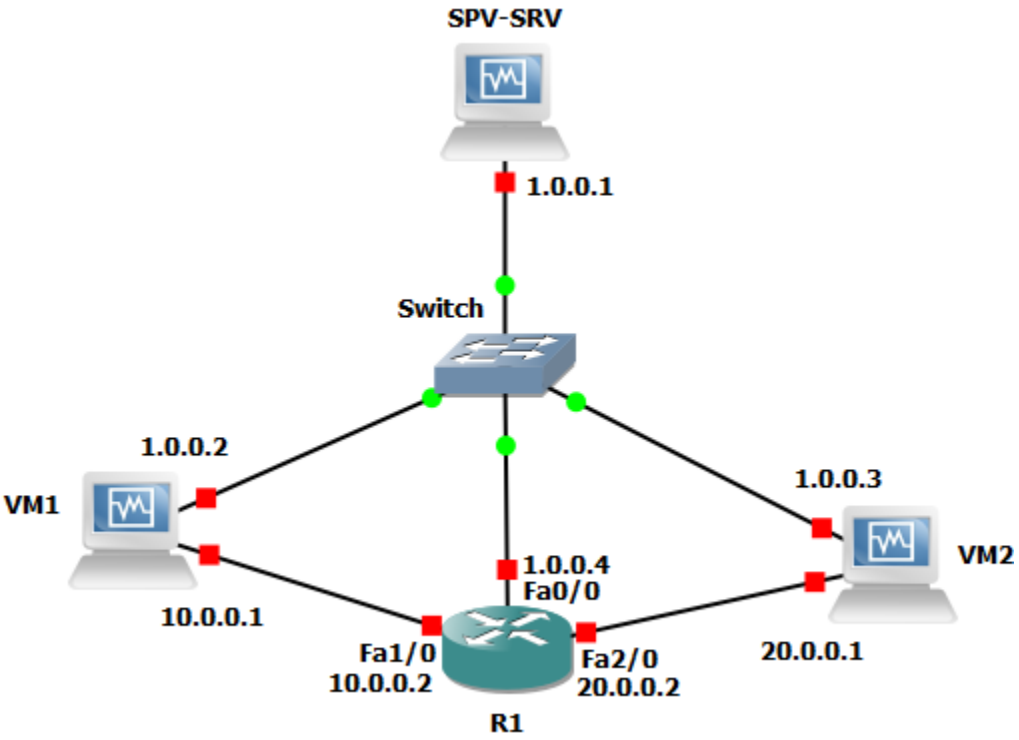

Figure 1: Architecture réseau simulée dans GNS3

- **VM1** : Machine virtuelle connectée au routeur via l'interface `FastEthernet1/0`
- **R1** : Routeur Cisco 3600 virtualisé
- **VM2** : Machine virtuelle connectée au routeur via `FastEthernet2/0`
- **SPV-SRV** : Station de supervision, connectée via un switch à l'interface `FastEthernet0/0` du routeur et aux VMs

3.1.1 Configuration du routeur Cisco

Voici les extraits de configuration essentiels appliqués au routeur Cisco 3600 (R1) :

```
interface FastEthernet0/0
 ip address 1.0.0.4 255.255.255.0
```

```
interface FastEthernet1/0
 ip address 10.0.0.2 255.255.255.0
 ip flow ingress
 ip flow egress
 ip route-cache flow

interface FastEthernet2/0
 ip address 20.0.0.2 255.255.255.0
 ip flow ingress
 ip flow egress
 ip route-cache flow

ip flow-export source FastEthernet0/0
ip flow-export version 5
ip flow-export destination 1.0.0.1 9999

snmp-server community public RO
snmp-server enable traps
```

Cette configuration permet à la station de supervision de recevoir les enregistrements NetFlow, les *traps* SNMP et pouvoir faire du *polling* de métriques SNMP sur le routeur.

3.1.2 Configuration SNMP sur les machines virtuelles

Pour permettre le *polling* SNMP depuis la station de supervision, les VMs doivent exécuter un agent SNMP. Voici les étapes de configuration sur Ubuntu :

```
# Installation de l'agent SNMP
sudo apt update
sudo apt install snmpd

# Édition du fichier de configuration
sudo nano /etc/snmp/snmpd.conf
```

Modifications à effectuer dans `snmpd.conf` :

```
agentAddress  udp:161

#Accès en lecture à toutes les métriques
view all included .1

# Restreindre au sous-réseau de supervision
rocommunity public 1.0.0.0/24 -V all
```

Puis redémarrer le service :

```
sudo systemctl restart snmpd
```

Cela permet à la station SPV-SRV d'interroger en SNMP les équipements du réseau.

3.2 Collecte et simulation des données

Pour évaluer les capacités de détection et de classification du système, nous avons développé plusieurs scripts Python spécialisés. Ils ont été déployés sur la station de supervision (SPV-SRV) afin de collecter les données télémétriques provenant des équipements, et de générer des anomalies réseau de manière contrôlée.

3.2.1 Collecte par SNMP polling

Le script `snmp_poller.py`[3] interroge régulièrement les équipements via SNMPv2c pour extraire des métriques système et réseau. Il utilise la bibliothèque `pysnmp` pour interroger différents OIDs spécifiques à chaque type d'équipement (Linux ou Cisco). Les principales métriques récupérées incluent :

- **CPU** : charge système à 1, 5 et 15 minutes, taux d'occupation utilisateur et système.
- **Mémoire** : quantité totale, utilisée, libre, swap disponible.
- **Réseau** : nombre d'octets reçus/envoyés (calcul des débits).

Les résultats sont stockés dans le fichier `snmp_poll.csv` sous forme structurée (timestamp, source, métrique, valeur, étiquette éventuelle, message).

3.2.2 Collecte des SNMP traps

Le script `snmp_trap_listener.py`[4] utilise la même bibliothèque pour mettre en place un agent passif qui écoute sur le port UDP 162. Il reçoit les traps envoyés automatiquement par le routeur Cisco (ex. changement d'état d'interface, redémarrage, incident matériel).
Chaque trap est horodaté et stocké dans `snmp_traps.csv`, incluant l'ensemble des OIDs présents dans le message.

3.2.3 Collecte des flux NetFlow

Le script `netflow_collector.py`[2] met en place un serveur UDP sur le port 9999 pour collecter les exports NetFlow v5 envoyés par le routeur. Il parse les paquets pour extraire les informations suivantes :

- **Adresse source / destination** et ports
- **Protocole**, volume total échangé (octets), nombre de paquets

Chaque flux est annoté automatiquement si certaines conditions sont détectées, par exemple :

- `HIGH_FLOW` : flux volumineux (plusieurs Mo en quelques secondes)
- `SUSPICIOUS_PORT` : ports cibles connus comme sensibles (23, 445, etc.)
- `ICMP_FLOW` : trafic ICMP massif

Les résultats sont enregistrés dans `netflow_flows.csv`.

3.2.4 Simulation d'anomalies réseau

Le script `anomaly_simulator.py`[1] automatise la génération d'événements perturbateurs dans le réseau afin d'enrichir le jeu de données avec des cas non triviaux. Il simule deux grandes familles d'anomalies :

- **Anomalies applicatives** sur les VMs (via SSH) :
  - **Surcharge CPU** avec `stress -c`
  - **Saturation mémoire** avec `stress -vm`
  - **Débit réseau élevé** avec `iperf3`
  - **Ping** entre VMs
- **Anomalies réseau** sur le routeur (via Telnet) :
  - **Coupure et réactivation d'une interface** (LinkDown/Up)

Le script fonctionne par cycles aléatoires : il enchaîne des événements sur les VMs, provoque une coupure de lien (ex. `shutdown Fa1/0`), attend un délai, puis enchaîne de nouveaux événements. Ce comportement permet de reproduire une dynamique réseau instable ou dégradée.

L'ensemble des scripts est présenté dans les annexes et permet de générer un jeu de données riche, hétérogène et annoté.

3.2.5 Extraits du dataset généré

Les scripts de collecte (polling SNMP, écoute de traps, collecte NetFlow) produisent des fichiers CSV structurés contenant les données de télémétrie et leurs éventuelles étiquettes. Voici quelques extraits caractéristiques.

Extrait typique de `snmp_poll.csv` (polling SNMP)

```
timestamp,source,record_type,metric,value,label,message
2025-04-03 22:55:08,VM1,SNMP_POLL,sysUpTime,587711.0,,
2025-04-03 22:55:08,VM1,SNMP_POLL,load1,0.0,,
2025-04-03 22:55:08,VM1,SNMP_POLL,load5,0.0,,
2025-04-03 22:55:08,VM1,SNMP_POLL,cpuUsage,0.0,,
2025-04-03 22:55:08,VM1,SNMP_POLL,memUsage,72.24,,
2025-04-03 22:55:08,VM1,SNMP_POLL,tx_rate,2142507.6,HIGH_TX,
2025-04-03 22:55:08,VM2,SNMP_POLL,rx_rate,3622908.8,HIGH_RX,
2025-04-03 22:55:08,R1,SNMP_POLL,memPoolFree,114331304.0,,
```

Ces lignes illustrent un sondage simultané sur les trois équipements (VM1, VM2, R1). On y observe des surcharges réseau détectées automatiquement grâce à l'étiquetage `HIGH_TX` et `HIGH_RX`.

Extrait de `snmp_traps.csv` (trap Cisco sur événement réseau)

```
timestamp,source,record_type,metric,value,label,message
2025-03-31 10:07:26,UNKNOWN,SNMP_TRAP,trap,,,
1.3.6.1.2.1.1.3.0 = 54148 |
1.3.6.1.6.3.1.1.4.1.0 = 1.3.6.1.4.1.9.9.41.2.0.1 |
1.3.6.1.4.1.9.9.41.1.2.3.1.5.4 = Interface FastEthernet2/0, changed state to up
```

Les traps SNMP sont capturés en temps réel et stockés avec leur contenu brut. Ici, une interface réseau a changé d'état à la suite d'un événement simulé.

Extrait de `netflow_flows.csv` (flux suspect détecté par NetFlow)

```
timestamp,source,record_type,metric,value,label,message
2025-03-31 10:47:50,1.0.0.4,NETFLOW,flow,122085,HIGH_FLOW,
Flow 1/2, Ver:5, Src:20.0.0.1:47610 -> 10.0.0.1:5201,
Pkts:122085, Bytes:183123201, Proto:6 | Large flow detected
2025-03-31 10:59:37,1.0.0.4,NETFLOW,flow,82,ICMP_FLOW,
Flow 2/4, Ver:5, Src:20.0.0.1:0 -> 10.0.0.1:2048, Pkts:82,
Bytes:6888, Proto:1 | ICMP flow flagged
```

Ces flux réseau sont étiquetés automatiquement en fonction de seuils critiques ou de comportements typiques (flux ICMP intensifs, transferts massifs sur ports suspects, etc.).

3.3 Protocole expérimental

Préparation des données Les données issues des fichiers CSV (`snmp_poll.csv`, `snmp_traps.csv`, `netflow_flows.csv`) ont été transformées et fusionnées dans un fichier au format `.arff`, standard pour Weka. Chaque enregistrement comporte les champs suivants :

- **timestamp** : horodatage de la mesure
- **source** : équipement concerné (VM1, VM2, R1)
- **record_type** : type de donnée (SNMP_POLL, SNMP_TRAP, NETFLOW)
- **metric** : nom de la métrique mesurée
- **value** : valeur brute ou calculée
- **label** : classe attribuée (NORMAL, CPU_OVERLOAD, HIGH_FLOW, etc.)

Extrait du fichier .arff

```
'2025-03-28 18:10:22','VM1','SNMP_POLL','rx_rate',0.0,'NORMAL'
'2025-03-28 18:10:22','VM1','SNMP_POLL','swapAvail',2044244.0,'NORMAL'
'2025-03-28 18:10:22','VM1','SNMP_POLL','swapTotal',2097148.0,'NORMAL'
'2025-03-28 18:10:22','VM1','SNMP_POLL','memFree',3337836.0,'NORMAL'
'2025-03-28 18:10:22','VM1','SNMP_POLL','tx_rate',0.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','cpuIdle',0.0,'CPU_OVERLOAD'
'2025-03-28 18:10:35','VM2','SNMP_POLL','swapTotal',2097148.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','memTotal',2018908.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','memUsage',47.03255423228795,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','memUsed',949544.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','memFree',3046692.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','swapAvail',2097148.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','load5',1.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','tx_rate',0.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','sysUpTime',807338.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','load1',0.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','load15',1.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','memPoolFree',114349540.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','cpuUsage',100.0,'CPU_OVERLOAD'
'2025-03-28 18:10:35','VM2','SNMP_POLL','rx_rate',0.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','cpuSystem',57.0,'NORMAL'
'2025-03-28 18:10:35','R1','SNMP_POLL','memPoolUsed',11732956.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','cpuUsage',10.0,'NORMAL'
'2025-03-28 18:10:35','VM2','SNMP_POLL','cpuUser',10.0,'NORMAL'
```

Méthodologie de classification Nous avons opté pour une approche d'apprentissage supervisé. La classe cible est la colonne `label`. Les algorithmes suivants ont été sélectionnés et testés dans Weka :

- `BayesNet`
- `NaiveBayes`
- `NaiveBayesUpdateable`
- `LibLINEAR`
- `LibSVM`
- `MultilayerPerceptron`
- `IBk`
- `LocalKNN`
- `FURIA`
- `NNge`
- `PART`
- `J48`
- `RandomForest`
- `RandomTree`
- `SimpleCart`
- `RepTREE`

Validation croisée Pour garantir une évaluation rigoureuse et éviter les biais de partition, une validation croisée à 10 plis (10-fold cross-validation) a été utilisée.

3.4 Analyse des performances

Pour chaque algorithme, Weka fournit une série de mesures quantitatives que nous avons exploitées pour évaluer la qualité des prédictions.

Métriques analysées

- **Kappa statistic** : accord entre la prédiction et les vraies classes
- **Accuracy (précision globale)**
- **Precision / Recall / F-Measure** (par classe)
- **False Positive / False Negative Rates**
- **ROC Area / PRC Area**
- **MAE, RMSE** : erreurs absolue et quadratique moyennes
- **RAE / RRSE** : erreurs relatives normalisées
- **Temps d'apprentissage**

4 Conclusion et perspectives

References

[1] ABHEMA. anomaly_simulator.py – learning nms project. https://github.com/ABHEMA/PRFE-Learning-NMS/blob/main/utils/anomaly_simulator.py, Avril 2025.

[2] ABHEMA. netflow_collector.py – learning nms project. https://github.com/ABHEMA/PRFE-Learning-NMS/blob/main/src/netflow_collector.py, Avril 2025.

[3] ABHEMA. snmp_poller.py – learning nms project. https://github.com/ABHEMA/PRFE-Learning-NMS/blob/main/src/snmp_poller.py, Avril 2025.

[4] ABHEMA. snmp_trap_listener.py – learning nms project. https://github.com/ABHEMA/PRFE-Learning-NMS/blob/main/src/snmp_trap_listener.py, Avril 2025.

[5] Ghazi Al-Naymat, Mouhammd Al-Kasassbeh, and Eshraq Al-Hawari. Exploiting snmp-mib data to detect network anomalies using machine learning techniques. *arXiv preprint arXiv:1809.02611*, September 2018.

[6] Ghazi Al-Naymat, Hanan Hussain, Mouhammd Al-Kasassbeh, and Nidal Al-Dmour. Accurate detection of network anomalies within snmp-mib data set using deep learning. *International Journal of Computer Applications in Technology*, 66(1):74–85, November 2021.

[7] Abul Bashar, Gerard Parr, Sally McClean, Bryan Scotney, and Detlef Nauck. Application of bayesian networks for autonomic network management. *Journal of Network and Systems Management*, 22(2):174–207, April 2014.

[8] Igor Fosić, Drago Žagar, Krešimir Grgić, and Višnja Križanović. Anomaly detection in netflow network traffic using supervised machine learning algorithms. *SSRN Electronic Journal*, November 2022.

[9] Abdelrahman Manna and Mouhammd Alkasassbeh. Detecting network anomalies using machine learning and snmp-mib dataset with ip group. *arXiv preprint arXiv:1906.00863*, June 2019.

[10] Sasha Velednitsky. The future of network monitoring: How ai and machine learning are changing the game. *NetFlow Logic Blog*, February 2025.