NeCSTGen : Génération de Trafic Réseau Réaliste par Apprentissage Profond

Fabien Meslet-Millet, Sandrine Mouysset, Emmanuel Chaput

NeCSTGen : Génération de Trafic Réseau Réaliste par Apprentissage Profond.

Fabien Meslet-Millet et Sandrine Mouysset et Emmanuel Chaput

IRIT, Université de Toulouse, France

La génération de trafic est un outil important pour de nombreuses activités liées aux réseaux de données, telles que la simulation, la planification et le provisionnement. Une telle génération pourrait être basée sur des modèles théoriques lorsqu'ils sont disponibles ; cependant, nous pensons que l'imitation du trafic précédemment collecté est une solution beaucoup plus générique. Les outils d'apprentissage profond sont désormais suffisamment matures pour fournir les bons outils pour capturer et reproduire les caractéristiques multi-échelles du trafic réseau.

Nous proposons une nouvelle architecture, appelée Network Clustering Sequential Traffic Generation (NeCSTGen), basée sur des modèles d'apprentissage profond. NeCSTGen peut générer un trafic réseau qui reproduit fidèlement le comportement original au niveau des paquets, des flux et des agrégats. Notre approche permet une compréhension et une génération innovante à grain fin sans avoir besoin de comprendre, en profondeur, le fonctionnement du protocole à générer. Notre architecture reproductible génère des données qui peuvent être exportées dans un fichier .pcap.

Mots-clefs : Apprentissage Profond, VAE, RNN, GMM, Génération de Trafic Réseau

1 Introduction

Selon [MMS13], il existe cinq types de générateurs traditionnels différents dont [TVIB05, SKB04, RRBB08, AS10, KRLM08, KLL+12]. Les générateurs traditionnels utilisent des distributions prédéfinies pour la génération ou des approches d'estimation statistique [VV09]. Le trafic ne correspondant pas à des distributions connues est plus difficile à générer.

Face aux limites des générateurs traditionnels, des méthodes de d'apprentissage profond ont été utilisées [VBN17, HSF19, SBJ+20, WLY+20, LKO20, ZN20, che21]. Elles fonctionnent mieux que les approches traditionnelles pour l'estimation statistique. Cependant, la génération de très longues séquences de paquets n'est pas possible avec les architectures d'apprentissage profond séquentiel actuelles. De plus, aucune approche n'est capable de traiter différents types de protocoles ou de dynamiques sur les données à générer. Enfin, il n'est pas possible de générer des comportements spécifiques dans le réseau.

Dans ce papier, nous proposons NeCSTGen une approche adaptative permettant de travailler avec tous les types de trafic, sans connaissance préalable. Nous fournissons également le code source [†] pour une recherche reproductible et nous montrons qu'il est possible de générer des fichiers .pcap réalistes qui peuvent être exploités pour différents cadres d'application.

2 Formalisation

Nous définissons trois niveaux de génération : (i) **Niveau paquet :** P l'ensemble des paquets générés, noté p_n, appartenant au même protocole applicatif. (ii) **Niveau flux :** f un flux individuel composé d'un ensemble de paquets p_n^j, avec j la position des paquets dans le flux. (iii) **Niveau agrégat :** A l'agrégat des flux générés par une application, utilisateurs .... Chaque niveau est généré indépendamment.

Pour chaque élément (paquet, flux ou agrégat), nous avons attribué des caractéristiques extraites avec $F(x) \in \mathbb{R}^k$ où k dépend de la nature de x. L'objectif de NeCSTGen est de prendre en entrée un ensemble de paquets notés P_{input} et d'extraire des caractéristiques pour chaque élément notés $V_{input} = F(P_{input})$. NeCSTGen apprend sur $V_{input} = \{v_0, \ldots, v_n\}$ et génère $V_{ouput} = \{h_0, \ldots, h_n\}$ avec $n \in \mathbb{N}$ permettant de construire les paquets à générer.

†. Code source : `https://github.com/fmeslet/NeCSTGen`

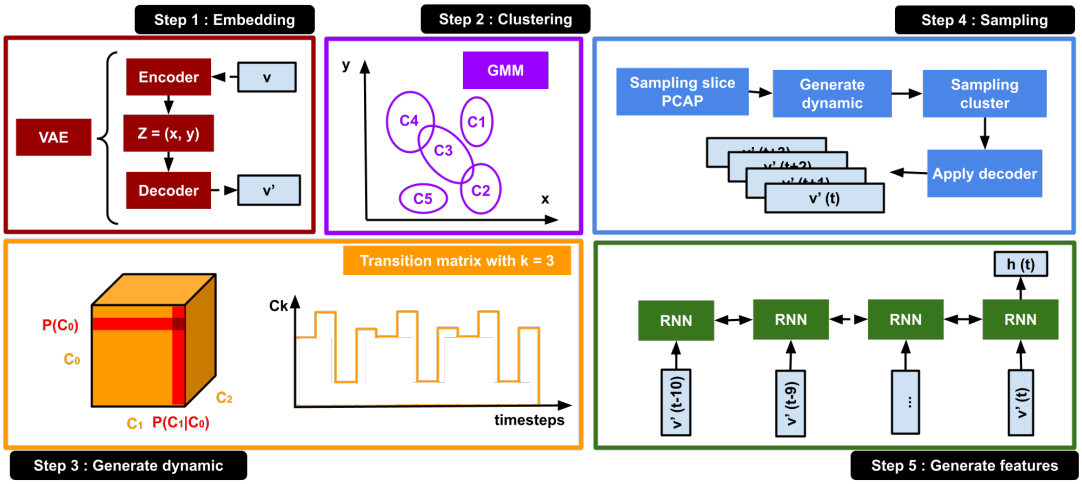

FIGURE 1 : Aperçu de l'architecture d'apprentissage profond, appelée NeCSTGen, pour la génération de trafic réseau.

3 Apprentissage profond

3.1 Génération de paquet

La figure 1 montre l'architecture proposée. Chaque vecteur de caractéristiques $v_n = F(p_n)$ est projeté dans un espace latent, noté Z, avec z_n les vecteurs projetés. Cette projection est réalisée à l'aide de la matrice E instanciée par l'encodeur du Variational Autoencoders (VAE) lors de l'étape 1. Un échantillon de l'espace latent Z, noté z_n, encapsule des informations sur un type de paquet (champs d'en-tête et statistiques) et sur le débit au moment de la capture (dynamique de haut niveau).

L'ensemble des vecteurs $\{z_0, \ldots, z_n\}$ qui appartiennent au même cluster, noté C_k, est identifié par le Gaussian Mixture Model (GMM) lors de l'étape 2. GMM identifie des groupes d'éléments similaires v_n sous forme de clusters, notés C_k avec $k \in \{0, ..., K\}$ où K est le nombre de clusters dans l'espace latent. Ces éléments peuvent être des paquets, des flux ou des éléments d'un agrégat. GMM associe à chaque cluster C_k une densité de probabilité gaussienne de paramètres $\alpha_k = (\mu_k, \Sigma_k)$, notée $P_k \sim N(\mu_k, \Sigma_k)$.

Ainsi, chaque cluster C_k identifie un ensemble de paquets ayant des caractéristiques similaires. Pour générer un vecteur de caractéristiques, on échantillonne un cluster C_k en utilisant la densité de probabilité associée P_k. Le $\widetilde{z_n} \sim P_k(\alpha_k)$ obtenu permet d'effectuer la transformation inverse en utilisant la matrice $D \approx E^{-1}$, instanciée par le décodeur VAE, pour obtenir les caractéristiques du cluster v'_n. Les informations contenues dans le vecteur v'_n sont suffisantes pour générer un paquet.

3.2 Génération de flux

Traité à l'étape 5, générer un flux f_n consiste à générer un ensemble de vecteurs ${v'_n}^0 \ldots {v'_n}^j$ vecteurs avec j la position du paquet associé p_n^j dans le flux. Cependant, les valeurs du vecteur doivent être cohérentes dans le temps. Pour cela, nous définissons $\{h_n^0 \ldots h_n^j\} = V_{output}$ les vecteurs de caractéristiques à générer afin d'obtenir un flux cohérent. En mode connecté, nous définissons un flux avec trois parties : un début, un milieu et une fin respectivement associés à un espace latent noté Z^B, Z^M et Z^E. En mode non connecté, un espace latent unique Z est suffisant. Pour chaque espace latent, nous trouvons un ensemble de clusters, notés $C_{i,k}^X$ avec : (i) $X = \{B, M, E\}$ la partie du flux, (ii) $i \in \mathbb{N}$ la position des paquets au sein du flux et $k \in \{0, \ldots, K\}$ le numéro du cluster. Ainsi, $\forall i \in \mathbb{N}, C_{i,k}^X \in Z_i^X$ et $X \in \{B, M, E\}$.

Générer un flux consiste à trouver les séquences de clusters $C_{i,k}^X$ à échantillonner afin de générer les caractéristiques de chaque paquet d'un flux. Ainsi, la probabilité d'échantillonner le cluster $C_{i,k}^X$ dépend des clusters échantillonnés précédant $C_{i-j,k}^X$ avec j le rang des clusters précédents. Cette tâche est réalisée par la matrice de transition lors de l'étape 3. Ainsi, $\forall X \in \{B, M, E\}, P(C_{i+1,k}^X) = P(C_{i,k}^X | C_{i-1,k}^X \ldots C_{i-j,k}^X)$ avec $z_n \in$

$C^X_{i+1,k}$ et $i, j \in \mathbb{N}$. Pour la génération, il suffit d'échantillonner la matrice de transition afin de générer une longue séquence de clusters.

Les clusters de la séquence générée sont échantillonnés pour obtenir les $\widetilde{z_n^{j-10}} \ldots \widetilde{z_n^j}$. Nous choisissons 10 arbitrairement. Ensuite, le décodeur D, issu du VAE, est utilisé pour obtenir les vecteurs ${v'_n}^{j-10} \ldots {v'_n}^j$. Enfin, avec le Recurrent Neural Network (RNN), on peut obtenir le vecteur h_n^j, cohérent avec le précédent vecteur généré. La génération avec le modèle $RNN({v'_n}^{j-10} \ldots {v'_n}^j) = h_n^j$ avec $n, j \in \mathbb{N}$. h_n^j contient les informations pour la construction d'un paquet p_n^j à l'intérieur du flux f_n. Par exemple, la différence de temps entre p_n^{j-1} et p_n^j.

4 Validation

4.1 Variables de dynamique

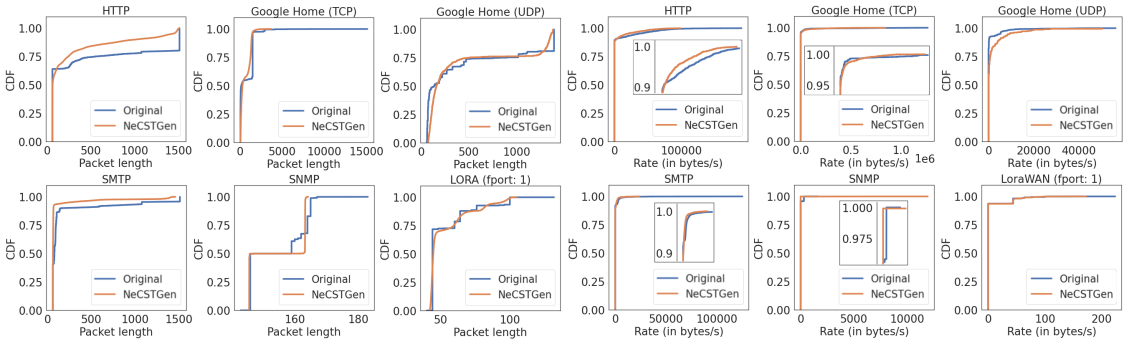

FIGURE 2 : Cumulative Distribution Function (CDF) des distributions de la taille des paquets (en octets) pour chaque protocole étudié (à gauche). CDF des distributions de débit en octets/s pour chaque protocole étudié (à droite).

La Figure 2 montre la CDF des différentes tailles de paquets en octets et des débits en octets/s entre les données originales et générées. Pour les débits, nous constatons de légers écarts (Simple Network Management Protocol (SNMP), Simple Mail Transfer Protocol (SMTP)). Cela s'expliquent par l'existence d'une longue période d'absence de communication ou d'une courte période de forte communication. Ces variations sont également présentes au sein des flux et agrégats générés. Nous utilisons une Logscale Diagram Estimate (LDE) pour effectuer une analyse par transformée en ondelettes discrètes. Une différence importante est observée pour SMTP, Hypertext Transfer Protocol (HTTP), LoraWAN (fport : 10) et Google Home. Si le trafic a beaucoup d'aléatoire ou des comportements différents, la dynamique globale n'est pas respectée. Les variations ou évènements rares (taille de paquets élevée) sont difficiles à appréhender par NeCSTGen. Les résultats obtenus sont similaires ou inférieurs à ceux observés dans la littérature pour les approches traditionnelles [SBJ+20, SKB04, RRBB08, VV09]. Ils sont supérieurs pour les approches par apprentissage profond [SBJ+20]. Ainsi, on perd en performance mais on gagne en adaptabilité. Par manque de place, les graphiques sont disponibles sur Github avec le code source.

4.2 Variables de champs

Nous avons évalué la génération de séquences de champs « flags » (par exemple : SYN-SYN/ACK-ACK). Nous constatons une différence moyenne d'environ 5% entre la proportion originale et celle générée pour le champs « flags ». Nous générons d'autres variables liées au protocole LoraWAN et à son environnement comme le Signal-to-Noise Ratio (SNR). L'évolution entre les variables SNR/Received Signal Strength Indication (RSSI) et SNR/Spreading Factor générées est cohérente avec les données originales. NeCSTGen est capable de générer des variables externes au protocole tout en gardant une cohérence globale avec les champs du protocole. A notre connaissance, aucune approche n'est capable de réaliser ce type de génération. Les graphiques sont disponibles sur Github avec le code source.

5 Conclusion

Nous proposons NeCSTGen, une architecture reproductible pour la génération de trafic réseau réaliste. Elle permet de générer des paquets, des flux et des agrégats, ainsi que des données externes sans aucune connaissance du protocole ou de la dynamique. A travers les résultats, nous avons montré la polyvalence de notre approche tout en gardant une cohérence globale au sein du trafic généré. Ainsi, NeCSTGen peut être utilisée pour différents cadres applicatifs, quel que soit le type de trafic et de protocoles. Dans une prochaine étape, nous envisageons d'améliorer notre approche afin de comprendre la variabilité du trafic liée à l'aspect humain lors de la génération.

Références

[AS10] Abdolreza Abhari and Mojgan Soraya. Workload generation for youtube. *Multimedia Tools Appl.*, 46(1), jan 2010.

[che21] Packet-Level Adversarial Network Traffic Crafting using Sequence Generative Adversarial Networks. March 2021. arXiv : 2103.04794.

[HSF19] Ramin Hasibi, Matin Shokri, and Mehdi Dehghan Takht Fooladi. Augmentation scheme for dealing with imbalanced network traffic classification using deep learning. *CoRR*, abs/1901.00204, 2019.

[KLL+12] Chia-Yu Ku, Ying-Dar Lin, Yuan-Cheng Lai, Pei-Hsuan Li, and Kate Ching-Ju Lin. Real traffic replay over WLAN with environment emulation. In *2012 IEEE WCNC*, Paris, France, April 2012. IEEE.

[KRLM08] Rachid El Abdouni Khayari, Matthias Rücker, Axel Lehmann, and Adisa Musovic. ParaSynTG : A Parameterized Synthetic Trace Generator for Representation of WWW Traffic. 2008.

[LKO20] Dongyang Li, Daisuke Kotani, and Yasuo Okabe. Improving Attack Detection Performance in NIDS Using GAN. In *2020 IEEE 44th COMPSAC*, Madrid, Spain, July 2020. IEEE.

[MMS13] Sandor Molnar, Peter Megyesi, and Geza Szabo. How to validate traffic generators ? In *2013 IEEE ICC*, Budapest, Hungary, June 2013. IEEE.

[RRBB08] Chloé Rolland, Julien Ridoux, Bruno Baynat, and Vincent Borrel. Using LiTGen, a realistic IP traffic model, to evaluate the impact of burstiness on performance. In *ICST*, Marseille, France, 2008. ICST.

[SBJ+20] Mustafizur R. Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Herve Debar. Generative Deep Learning for Internet of Things Network Traffic Generation. In *2020 IEEE PRDC*, Perth, Australia, December 2020. IEEE.

[SKB04] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon : A flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev.*, 32(1), jun 2004.

[TVIB05] Tao Ye, D. Veitch, G. Iannaccone, and S. Bhattacharyya. Divide and Conquer : PC-Based Packet Trace Replay at OC-48 Speeds. In *TridentCom '08*, Trento, Italy, 2005. IEEE.

[VBN17] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. A Deep Learning Based Method for Handling Imbalanced Problem in Network Traffic Classification. In *SoICT 2017*, Nha Trang City Viet Nam, December 2017. ACM.

[VV09] Kashi Venkatesh Vishwanath and Amin Vahdat. Swing : Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3), 2009.

[WLY+20] Pan Wang, Shuhang Li, Feng Ye, Zixuan Wang, and Moxuan Zhang. Packetcgan : Exploratory study of class imbalance for encrypted traffic classification using cgan. In *ICC 2020*, 2020.

[ZN20] Pasquale Zingo and Andrew Novocin. Can GAN-Generated Network Traffic be used to Train Traffic Anomaly Classifiers ? In *2020 11th IEEE IEMCON*, Vancouver, BC, Canada, November 2020. IEEE.