Network Traffic Generation for Evaluation of Intrusion Detection Tools

Adrien Schoen

COLLEGE DOCTORAL BRETAGNE

MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

CentraleSupélec

THÈSE DE DOCTORAT

CENTRALESUPÉLEC

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Network Traffic Generation for Evaluation of Intrusion Detection Tools

Machine learning based generation of synthetic network flows

Par
Adrien SCHOEN

Thèse présentée et soutenue à Rennes, France, le December 18th, 2024

Numéro de thèse: 2024CSUP0020

Rapporteurs avant soutenance :

Jilles VREEKEN	tenured faculty, CISPA Helmholtz Center for Information Security
Herve DEBAR	Professeur, Telecom SudParis

Composition du Jury :

Président :	Maryline LAURENT	Professeur, Telecom SudParis
Examinateurs :	Pierre-Henri WUILLEMIN	Maitre de conference, Sorbonne Université
	Ludovic ME	Chercheur Contractuel Sénior, INRIA
	Gregory BLANC	Maitre de conference, Telecom SudParis
	Frederic MAJORCZYK	Ingénieur, DGA-MI
	Yufei HAN	Chercheur Contractuel Sénior, INRIA
Dir. de thèse :	Ludovic ME	Chercheur Contractuel Sénior, INRIA

Invité(s) :

Pierre-Francois GIMENEZ	Chercheur contractuel, INRIA

Acknowledgements

First and foremost, profound gratitude is extended to God, Creator and Sustainer of all things, for the world He has forged—a world that sparks humanity's scientific curiosity and drives our pursuit of understanding. Scientific research, in my view, is rooted in reverence for this vast and wondrous universe, whose beauty and coherence provide endless inspiration.

I am deeply grateful to my thesis supervisor, Ludovic Mé, whose unwavering support, patience, and invaluable guidance have been central throughout these three years of research. His insightful feedback and encouragement to pursue rigor have been instrumental in shaping my approach to this work, providing skills that will no doubt serve me well in my career.

Sincere appreciation is also extended to my thesis committee, Pierre-François Gimenez, Gregory Blanc, Frédéric Majorczyk, and Yufei Han, whose expertise and ongoing support enriched this project tremendously. Their feedback, illuminating insights, and encouragement significantly enhanced the quality of this thesis.

Thanks go as well to the PIRAT research team for their warm welcome and to Dr. Vreeken and his team at CISPA, in particular Mr. Joscha Cueppers, for making my two-month stay in Saarbrücken a deeply rewarding experience. The collaborative environment, openness, and shared knowledge within these teams played a major role in advancing this work.

Heartfelt gratitude goes to my family, who have remained a steadfast source of support throughout this demanding journey. Despite the distance and long hours, their encouragement and patience have been a vital pillar of strength. This achievement is as much theirs as it is mine.

Appreciation is also extended to the Discord group Krak, whose camaraderie and barbecues brought invaluable moments of relief and laughter along the way.

Finally, a special mention goes to onche.org, which provided three years of laughter and levity. Warm thanks to @GrosMalin, @Ulrich-Amalric, @Orteils, @Leakytap, @Johndoe2424, @Orgetorix, @Aghar, @123pk, @Mao, @CleaMolette and @samlaitbrize for making this journey all the more memorable.

Résumé

1 Introduction

La cybersécurité est un domaine crucial en raison de la dépendance croissante de la société aux infrastructures numériques interconnectées, fréquemment ciblées par des cybermenaces sophistiquées. Les systèmes de détection d'intrusion réseau (NIDS, pour Network Intrusion Detection Systems) jouent un rôle clé dans la défense contre ces menaces en surveillant et en analysant le trafic réseau pour détecter des activités suspectes. Parmi ces systèmes, ceux basés sur la détection d'anomalies sont particulièrement adaptés pour identifier les menaces émergentes en détectant les écarts par rapport au comportement réseau normal. Cette thèse se concentre sur ces approches, et plus particulièrement sur celles qui s'appuient sur l'apprentissage automatique pour la détection d'anomalies. Ces méthodes reposent sur des jeux de données de haute qualité, indispensables à l'entraînement des modèles et à l'évaluation de leurs performances.

Cependant, obtenir de tels jeux de données présente plusieurs défis. Le trafic réseau réel est difficile à exploiter en raison de réglementations strictes sur la confidentialité, comme le RGPD (pour Règlement Général sur la Protection des Données), qui limitent l'accès aux données sensibles. Même lorsqu'il est disponible, l'étiquetage du trafic réel est fastidieux et sujet à des erreurs. De plus, ces données peinent souvent à refléter la diversité nécessaire pour modéliser fidèlement les comportements réseau, qu'ils soient normaux ou liés à des scénarios d'attaque. La nature dynamique des réseaux modernes complique davantage la tâche de constituer des jeux de données exhaustifs et représentatifs.

Pour pallier ces limitations, les simulations constituent une solution couramment utilisée. Elles permettent de modéliser des scénarios spécifiques en définissant des paramètres comme la topologie du réseau, les types de trafic, et les comportements des hôtes. Cette flexibilité explique leur popularité dans les domaines académiques et industriels, notamment pour l'évaluation des NIDS. Cependant, les simulations présentent également des limites importantes. Elles sont souvent fortement contraintes par les hypothèses initiales et les configurations définies, ce qui réduit leur capacité à s'adapter à des scénarios complexes ou évolutifs. De plus, à mesure que la taille et la complexité du réseau simulé augmentent, les besoins en ressources computationnelles croissent proportionnellement, chaque nouveau comportement

simulé nécessitant une machine supplémentaire. Par ailleurs, le temps de simulation est directement lié à la durée des jeux de données à produire, ce qui peut entraîner des délais importants pour générer des volumes de trafic exploitables. Cette double contrainte rend les simulations à grande échelle difficilement viables.

Face à ces limites, la génération de trafic réseau synthétique émerge comme une alternative prometteuse. Cette approche repose sur l'apprentissage de modèles à partir de données réelles, permettant d'extraire leurs caractéristiques essentielles, telles que les distributions statistiques et les relations entre variables. Une fois ces modèles entrainés, ils sont capables de générer directement un nouveau trafic complètement synthétique, contournant de ce fait les contraintes liées à la confidentialité des données sensibles.

Cependant, bien que la génération de trafic synthétique présente des avantages théoriques prometteurs par rapport aux simulations, son applicabilité pratique reste encore peu explorée. Cette thèse a pour objectif principal de vérifier si ces avantages se traduisent effectivement dans des contextes réels ou si la génération synthétique introduit des limitations ou des problèmes supplémentaires. Dans ce cadre, nous avons développé une méthode pratique et nous avons analysé son potentiel à remplacer les simulations dans l'évaluation des NIDS. Nos travaux se sont concentrés sur la génération de flux réseau bénins, en écartant volontairement les activités malveillantes.

2 Limitations des solutions existantes pour la génération de flux bénins synthétiques

2.1 Approches basées sur les GAN

Les réseaux antagonistes génératifs (GAN, pour Generative Adversarial Networks) se sont largement imposés pour leur capacité à générer des données synthétiques réalistes en apprenant les schémas complexes présents dans le trafic réel. Bien qu'ils soient prometteurs dans la reproduction des distributions statistiques, les GAN sont extrêmement coûteux en terme de calcul, rendant leur application à grande échelle difficile. En outre, ils rencontrent des difficultés pour modéliser les dépendances entre différentes variables, un élément crucial pour produire un trafic réseau fidèle et cohérent.

Plusieurs travaux ont tenté d'améliorer ces limites des GAN, notamment en introduisant des mécanismes pour capturer les dépendances temporelles dans le trafic réseau. Cependant, malgré ces efforts, ces approches échouent souvent à modéliser pleinement les dépendances

temporelles inhérentes au trafic réseau. Ces dépendances représentent les relations chronologiques entre événements, comme l'ordre des flux, et sont essentielles pour refléter la nature dynamique des réseaux modernes. Par exemple, une requête DNS précédant une requête HTTP vers un serveur externe est une séquence typique qui illustre l'importance des dépendances temporelles pour modéliser le trafic réseau. Sans une prise en compte adéquate de ces relations, le trafic synthétique manquera de cohérence et de réalisme.

2.2 Problèmes dans l'évaluation du trafic synthétique

Un obstacle majeur dans l'adoption du trafic synthétique comme alternative aux simulations est lié à son évaluation. Les benchmarks actuels s'appuient souvent sur des métriques liées à l'utilité pratique, comme l'amélioration des performances des NIDS. Si ces indicateurs sont utiles, ils tendent à dépendre fortement des scénarios pour lesquels ils sont conçus, limitant ainsi leur portée générale. Cette approche réduite empêche une évaluation complète et rigoureuse de la qualité des jeux de données.

Un autre aspect souvent négligé dans l'évaluation du trafic synthétique est la capacité des modèles génératifs à produire des motifs novateurs, c'est-à-dire des schémas qui ne se limitent pas à reproduire simplement les données d'entraînement. Ne pas évaluer cet aspect augmente le risque de surapprentissage, une situation où les données générées par le modèle ne sont que des copies de ses données d'entrainement.

Enfin, la plupart des frameworks d'évaluation n'intègrent pas une mesure de la préservation des dépendances temporelles. Or, le trafic réseau est par essence temporel, avec des flux et des événements structurés selon des chronologies précises. En l'absence de telles analyses, même si le trafic généré peut paraître cohérent, il peut échouer à reproduire fidèlement les dynamiques temporelles caractéristiques des réseaux réels.

3 Contributions de cette thèse

3.1 Un cadre d'évaluation unifié

Cette thèse propose un benchmark complet permettant d'évaluer la qualité du trafic synthétique selon quatre critères principaux : le réalisme, la diversité, la nouveauté et la conformité aux spécifications des protocoles.

Le réalisme garantit que les propriétés statistiques du trafic généré correspondent étroitement à celles du trafic réseau réel, en termes de distributions de variables comme les tailles

de paquets ou les intervalles entre les flux. La diversité mesure l'étendue des comportements capturés dans les données synthétiques, reflétant la capacité à représenter une grande variété de scénarios et d'environnements réseau. La nouveauté évalue si le trafic généré est capable d'introduire des schémas nouveaux qui ne reproduisent pas simplement les données d'entraînement. Enfin, la conformité vérifie que le trafic respecte les normes des protocoles réseau et les spécifications du domaine, garantissant son utilité dans des contextes réels.

Chacun de ces critères est évalué à l'aide de plusieurs métriques spécifiques. Par exemple, pour le réalisme, la distance de Wasserstein a été utilisée pour comparer les distributions des tailles de paquets et des intervalles entre paquets générés à celles des données réelles. Pour la diversité, plusieurs métriques sont utilisées pour mesurer la part de la base de données réelles recouverte par la base des données générées, comme la couverture. La conformité est évaluée par une succession de tests de validité vis-à-vis de différents protocoles réseaux. Enfin, la nouveauté est mesurée en analysant la résistance du jeu de données généré à des attaques par inférence d'appartenance.

Ce benchmark offre également des méthodes spécifiques pour analyser la préservation des dépendances temporelles, un aspect essentiel souvent sous-estimé dans les approches existantes. Pour les dépendances temporelles, une des métriques utilisées est la différence entre les fonctions d'autocorrélation des jeux de données réels et générés. Cette approche permet d'évaluer dans quelle mesure les relations temporelles entre événements sont préservées dans le trafic synthétique. En garantissant que le trafic généré reproduit fidèlement les séquences et les dynamiques des scénarios réels, ce benchmark fournit un outil essentiel pour comparer les différentes méthodes de génération de trafic.

3.2 Utilisation des réseaux bayésiens

Afin de résoudre les problèmes liés à la modélisation des dépendances entre variables, cette thèse propose d'utiliser les réseaux bayésiens (BN, pour Bayesian Networks) pour la génération de flux individuels. Les BN permettent de représenter efficacement les relations conditionnelles entre variables, offrant ainsi une solution adaptée pour générer un trafic réaliste tout en réduisant les coûts computationnels par rapport aux GAN.

Une attention particulière a été accordée à la gestion des variables à forte cardinalité et à la discrétisation des variables continues. Pour les variables à forte cardinalité, les adresses IP et les ports éphémères sont regroupés en catégories pertinentes afin de préserver les relations essentielles tout en simplifiant la modélisation. Pour la discrétisation des variables continues, deux méthodes ont été utilisées : une approche basée sur les quantiles, qui divise la plage de

valeurs de chaque variable en intervalles égaux en fonction de leur distribution, et une autre utilisant des modèles de mélange gaussien (Gaussian Mixture Models) qui partitionnent cette plage en plusieurs noyaux de gaussiennes aux paramètres ajustables.

Les résultats expérimentaux, menés à l'aide du benchmark présenté précédemment, montrent que les réseaux bayésiens surpassent les GAN en termes de réalisme et de diversité des flux générés. Ces comparaisons ont été réalisées sur les jeux de données UGR'16 et CIC-IDS 2017. Toutefois, leur performance peut varier en fonction de la complexité des jeux de données, avec un avantage plus prononcé sur des jeux de données composés de peu de variables.

3.3 *FlowChronicle* : Capture des dépendances temporelles

Pour étendre l'utilisation des BN à la capture des dépendances temporelles, cette thèse présente *FlowChronicle*, un cadre combinant les BN avec une méthode d'extraction de motifs basée sur le principe de la Longueur Minimum de Description (MDL pour Minimum Description Length). Cette approche identifie et encode les schémas récurrents dans les flux réseau, permettant de générer des séquences cohérentes qui reproduisent les dynamiques temporelles des réseaux.

Concrètement, *FlowChronicle* décompose les flux en tableaux compacts comprenant des cellules fixes (valeurs constantes), des cellules réutilisées (relations entre flux) et des cellules libres (valeurs à générer). Par exemple, un motif peut représenter une requête DNS suivie d'un accès à un serveur externe, tout en intégrant une variabilité dans les tailles des paquets.

Des experiences menées sur le jeu de données CIDDS-001 montrent que *FlowChronicle* préserve efficacement les dépendances temporelles tout en surpassant significativement les autres méthodes de génération synthétique, montrant sa supériorité pour produire un trafic qui reflète fidèlement la dynamique des réseaux réels. Ici aussi, la comparaison a été rendue possible par notre benchmark, assurant une évaluation rigoureuse des performances. De plus, les motifs détectés par *FlowChronicle* sont analisables par un expert réseau, rendant notre génération en partie explicable.

4 Conclusion

Cette thèse a permis de combler plusieurs lacunes des approches existantes en génération de trafic synthétique. En introduisant les réseaux bayésiens pour la modélisation des flux et *FlowChronicle* pour capturer les dépendances temporelles, ces travaux ont jeté les bases d'une génération de trafic plus réaliste et utilisable dans des contextes réels. Le framework

d'évaluation unifié présenté ici constitue un outil essentiel pour comparer objectivement les différentes méthodes de génération, tout en mettant en avant des critères cruciaux comme la préservation des dépendances temporelles. Toutes ces avancées ouvrent de nouvelles perspectives pour une meilleure évaluation des NIDS, en réduisant leur dépendance aux simulations traditionnelles.

Les recherches futures devraient explorer l'application de ces méthodes à des scénarios plus complexes et étendre leur utilisabilité à des environnements réels. Ces avancées pourraient être directement utilisées pour évaluer des NIDS dans des environnements de production, en particulier dans des secteurs comme les réseaux industriels ou les systèmes IoT, où les contraintes de ressources et les risques liés à la confidentialité sont particulièrement élevés. L'intégration de modèles de langage de grande taille (LLM pour Large Language Model) pourrait enrichir la qualité et la diversité du trafic généré. Enfin, des travaux devront également être menés pour garantir que les méthodes respectent les exigences en matière de confidentialité, notamment en réponse à des régulations comme le RGPD.

Table of Contents

Chapter 1

Introduction

1.1 Global Context

As the world becomes increasingly interconnected through digital infrastructures, the importance of cybersecurity has grown substantially. The global digital economy heavily relies on secure communication channels, making cybersecurity a critical concern for governments, businesses, and individuals. This is reflected in the projected economic impact of cybercrime, which is expected to cost the world 10.5 trillion dollars annually by 2025, up from 3 trillion dollars in 2015[1].

Cyber threats are becoming increasingly sophisticated, with attackers employing advanced techniques to breach security defenses. These threats encompass many activities, including data breaches, ransomware attacks, and Denial of Service attacks. For instance, according to Verizon, ransomware attacks have doubled in frequency over the past two years, underscoring the evolving nature of these threats[2].

In this evolving threat landscape, protecting network communications has become critical. Network-based attacks have the potential to compromise sensitive data, disrupt essential services, and cause significant financial and reputational damage. As a result, securing network communications has become one of the top priorities for organizations worldwide, especially in critical infrastructure sectors such as finance, healthcare, and energy[3].

To detect and mitigate threats in network communications, **N**etwork **I**ntrusion **D**etection **S**ystems (**NIDS**) have been developed as critical tools for cybersecurity. NIDS monitor network traffic in real time, analyzing data packets to identify suspicious activities that may indicate a security breach. These systems inspect the traffic to detect known attack signatures or abnormal behaviors that deviate from established baselines [1]. A well-known example of an NIDS

[1] Cybersecurity Ventures, "Cybercrime To Cost The World 10.5 Trillion Dollars Annually By 2025", 2020. Available: https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/

[2] Verizon, "2023 Data Breach Investigations Report (DBIR)", 2023. Available: https://www.verizon.com/business/resources/reports/dbir/

[3] World Economic Forum "Global Cybersecurity Outlook 2023" 2023. Available: https://www.weforum.org/reports/global-cybersecurity-outlook-2023/

is Snort[4], an open-source tool that detects a wide range of attacks by analyzing network traffic in real-time. In [2], Martin Roesch, the creator of Snort, describes it as a lightweight and flexible NIDS capable of detecting various types of attacks, such as buffer overflows or stealth port scans.

While NIDS are effective at detecting some attacks, they often generate a large number of alerts, including a significant proportion of **false positives**—benign activities being incorrectly flagged as malicious. This flood of alerts contributes to **alert fatigue**, making it harder for security teams to distinguish genuine attacks from benign activities. The growing volume of alerts and a shortage of skilled cybersecurity professionals exacerbate these challenges, leading to inefficiencies in attack detection and response. Both researchers and industry experts have recognized this issue. For instance, Axelsson details the trade-offs between false positives and detection accuracy in NIDS [3]. Similarly, France's national cybersecurity strategy highlights the urgent need for more skilled professionals to address these inefficiencies[5]. At the same time, the European Union, through the European Union Agency for Cybersecurity (ENISA), has launched initiatives aimed at enhancing the skills of cybersecurity personnel across member states[6].

1.2 The need for synthetic network traffic

Evaluating NIDS' effectiveness is increasingly important due to the growing emphasis on data sovereignty in Europe, which demands locally developed and certified cybersecurity solutions[7]. This evaluation requires large datasets containing both benign and malicious network traffic to ensure accurate threat detection while minimizing false positives.

However, acquiring **benign traffic**—a key element for evaluating the false positive rate of NIDS—poses significant challenges. There are three primary ways to obtain network traffic for NIDS evaluation: recording it directly from network communications, simulating it in a controlled testbed environment, or generating it using modeling techniques. **Real traffic** refers to data recorded from actual network communications, whereas **simulated traffic** is created through controlled experiments in a testbed. **Synthetic traffic**, on the other hand, is generated

[4] https://www.snort.org/

[5] Ministère de l'Économie, "Stratégie nationale pour l'accélération de la cybersécurité", 2021. Available: https://www.economie.gouv.fr/strategie-nationale-acceleration-cybersecurite

[6] ENISA, "Education", 2023. Available: https://www.enisa.europa.eu/topics/education

[7] European Commission, "A European strategy for data", 2020. Available: https://ec.europa.eu/commission/presscorner/detail/en/fs_20_283

using algorithms that replicate the characteristics of real traffic. In the following Subsections, we will examine these three methods in detail.

1.2.1 Challenges with real network traffic

Acquiring real-world benign traffic for NIDS evaluation often involves capturing sensitive and personal information, raising significant ethical and legal concerns. Privacy laws, such as the General Data Protection Regulation (GDPR) in Europe, impose strict guidelines on collecting, storing, and sharing personal data. This makes it challenging to use real traffic for testing NIDS without violating privacy rights [4].

Moreover, accurately labeling real network traffic is a complex and time-consuming task. Expert knowledge is required to identify and categorize different network activities as benign or malicious. Mislabeling can result in flawed evaluations of NIDS, where benign traffic may be mistakenly classified as malicious or vice versa. The difficulty of obtaining reliably labeled data adds further complexity to the use of real network traffic for developing and testing NIDS [5].

1.2.2 Simulation as a possible solution

Researchers have turned to simulating traffic in controlled environments to overcome the challenges of using real network traffic. This involves generating traffic that mimics network activity without involving real human users. Instead, traffic is generated by network automata, such as web crawlers [6], email generators [7], or bots simulating user behavior based on predefined profiles [8]. This approach allows researchers to produce and share large datasets without the privacy concerns linked to real human traffic, as the virtual users involved are not real people. Additionally, because the behavior of simulated users is predefined, labeling the traffic is easier, which is a significant advantage over manually labeling real traffic [8].

However, traffic simulation still has limitations, particularly in terms of **adaptability** and **scalability**. Simulated traffic is tied to the specific behaviors and conditions set at the beginning of the simulation. Once the simulation starts, it follows these predefined conditions, making it challenging to adapt the traffic to new requirements, such as changes in the number of hosts, types of activities, or network topology [9]. This is a lack of *adaptability*. Furthermore, network simulation requires the actual execution of processes within the simulation environment, which becomes increasingly complex and costly as the number of hosts and the diversity of applications expand. This escalation in resource demands restricts the simulation's ability to model the full scope of network complexity and variability, thus limiting its utility in evaluating

NIDS under realistic conditions [6]. This is a lack of *scalability*.

In response to these limitations, researchers are increasingly turning to synthetic traffic generation as a more adaptable and scalable alternative to simulation.

1.2.3 Synthetic Traffic Generation

To address the limitations of both actual and simulated traffic, the research community has increasingly resorted to **synthetic traffic generation**, which leverages modeling algorithms to learn the characteristics of existing traffic and reproduce them [9]–[11]. These algorithms, often based on techniques such as machine learning or statistical models, enable the creation of synthetic traffic that closely mirrors real-world network activities without containing any much user information. Moreover, **synthetic traffic** can be generated with precise labeling, eliminating the need for the labor-intensive and error-prone process of manually labeling real traffic data.

Synthetic traffic generation addresses the *adaptability* limitations of traditional simulation by allowing for targeted data augmentation. Generative algorithms can produce additional samples of specific traffic classes, enriching datasets to improve NIDS performance without extensive manual intervention or rerunning entire simulation [11]. This adaptability is particularly valuable when dealing with imbalanced datasets, where certain types of network traffic are underrepresented.

Additionally, synthetic traffic generation overcomes the *scalability* limitations of simulation by decoupling the generation process from specific network components and configurations. Because it is based on an underlying modeling of the traffic itself, synthetic generation can scale easily with minimal computational cost, regardless of the number of hosts, network size, or activity diversity. This *scalability* could allow researchers to produce high volumes of complex traffic, reflecting real-world variability, without the prohibitive resource demands of traditional simulations[12].

Synthetic traffic generation also offers a significant advantage in terms of *speed*, enabling researchers to produce large datasets quickly without the constraints of real-time data capture or lengthy simulations. Unlike simulations, which are bound by the actual time required to replicate network activities, synthetic generation can create an hour's worth of traffic in a fraction of that time. This rapid generation process allows for swift adaptation of datasets to new testing requirements, making it particularly well-suited for NIDS evaluations in dynamic environments where conditions and security challenges frequently change [12].

However, despite the promising advantages of synthetic data generation, these expecta-

tions remain largely unproven in the context of network traffic data. While synthetic data generation has gained significant traction in other domains, such as image synthesis [13] and natural language generation [14], its application in the field of network security, particularly in the evaluation of NIDS, is still in its infancy. Since 2015, following the introduction of the PU-IDS dataset [15], there have been no significant advancements or widespread adoption of synthetic network traffic for NIDS evaluation or the creation of network datasets.

In the case of the PU-IDS dataset, Singh et al. try to replicate the NSL-KDD dataset's statistical characteristics by generating new traffic instances through a process that randomly samples from these statistical distributions. This approach, which seeks to construct a dataset based solely on abstracted statistical properties rather than on actual or simulated network activities, represents, to the best of our knowledge, the only known effort to employ synthetic traffic generation for creating and sharing a dataset.

Despite its potential, synthetic traffic generation comes with significant limitations. First, access to real traffic data is still required to build the initial model, which can be sensitive and restricted. The process of modeling itself can be time-consuming, especially for complex networks, and there is always a risk of overfitting or replicating the training data too closely, reducing the diversity of the generated traffic.

As a result, while synthetic traffic generation shows promise for faster, more adaptable, and scalable data production compared to traditional simulation methods, these advantages remain largely untested. More research and practical implementations are necessary to fully assess whether synthetic traffic can reliably replace simulated traffic to create comprehensive network datasets.

1.3 Problem statement

Our thesis contributes to the ongoing effort to replace simulation with synthetic traffic generation to create datasets of benign traffic. To this end, multiple generative models are being proposed. With the rise of generative AI, several network traffic generation methods emerge, utilizing Generative Adversarial Networks (GAN) [16], Variational Autoencoders [17], and Diffusion Models [18] among others [19]. Despite these advancements, synthetic traffic generation has not yet produced any widely adopted datasets for NIDS evaluation.

Furthermore, no standard benchmark is established for evaluating the quality of generated traffic, making any comparison between different models—or even between synthetic traffic generation and simulation—difficult. This lack of standardized evaluation metrics represents

a significant issue in current research.

1.3.1 Research objectives

In this thesis, we aim to develop a synthetic traffic generation system with the potential to replace simulation as the primary method for generating network traffic datasets for NIDS evaluation. To achieve this, we undertake several key steps.

First, we identify the limitations and challenges present in current synthetic traffic generation methods. While these methods show promise in other fields, they have not yet been able to supplant simulation in network traffic dataset creation. Understanding why these models cannot currently replace simulation is a critical step in our research.

Second, we propose a solution that addresses these identified limitations. Our synthetic traffic generation method overcomes the specific challenges that have hindered previous attempts, ensuring that it generates traffic with the accuracy, diversity, and scalability required for NIDS evaluation.

Third, after developing our solution, we evaluate and compare it against existing state-of-the-art synthetic traffic generation methods. The goal is not necessarily to outperform these models but to address the fundamental limitations that have hindered their adoption in place of simulation. By resolving these issues, we aim to make synthetic traffic generation methods as reliable as traditional simulation approaches. To facilitate this comparison, we introduce a benchmarking framework that provides standardized metrics and criteria for assessing the quality of generated traffic. Establishing this benchmark serves as a crucial contribution of this thesis, enabling consistent and reliable evaluation across different generation methods.

Fourth, to establish the viability of our synthetic traffic generation system as a replacement for simulation, we compare its performance directly against traditional simulation methods. This comparison is essential for assessing whether synthetic generation can produce datasets more efficiently while achieving the quality needed for accurate NIDS evaluation.

1.3.2 Assumptions and Scope of Our Study

To make our research more focused and manageable, we make several reasonable assumptions that help narrow the scope while allowing us to address the core challenges of network traffic generation.

First, we choose to focus specifically on the generation of **network flows**[8], as this represents

[8] Records of communication between two terminals in a network, see Section 2.1 for further details

a manageable and widely studied subset of network traffic. Network flow data is commonly used in NIDS because it balances granularity and scalability, making it suitable for large-scale network monitoring. For example, Sommer and Paxson (2010) highlight the widespread use of network flow data in NIDS research, noting that it is often preferred due to its ability to summarize communication between endpoints efficiently, enabling effective intrusion detection across large networks [20]. Additionally, as network traffic becomes increasingly encrypted, NIDS are often limited to analyzing flow data rather than the content of the traffic itself. This restriction makes network flows even more critical for modern intrusion detection systems. By concentrating on network flows, we can delve deeper into the specific challenges of generating this data type, providing more meaningful insights that directly benefit NIDS evaluation.

Next, we concentrate on **GAN** as the primary approach for synthetic traffic generation because they are by far the most widely used method in this field, as will be discussed in Section 3.1. By doing so, we can leverage the extensive research already available on GAN in other domains while addressing the specific obstacles related to their application in network traffic generation.

Finally, our study focuses solely on the generation of **benign traffic**, as our primary objective is to evaluate the false positive rate of NIDS. Generating malicious traffic is outside the scope of this work, as it presents its own set of challenges and is actively being explored in other research [21], [22]. Furthermore, combining generated benign traffic with malicious traffic presents its own set of challenges [23]. By restricting our study to benign traffic, we can more effectively address the core challenges of generating and evaluating traffic for false-positive rate assessment.

1.4 Contributions

This thesis makes several key contributions to the field of network traffic generation for NIDS evaluation:

- **Comprehensive Survey of Synthetic Traffic Generation Methods:** The first contribution of this thesis is a thorough survey of methods used for synthetic traffic generation presented in Section 3.1. This survey highlights the emerging trends and uncovers the limitations of existing approaches, offering a clear understanding of the current state of research and identifying gaps that need to be addressed.

- **Network Flow Generation using Bayesian Networks:** The second contribution is the in-

troduction of a novel synthetic network flow generation method based on Bayesian Networks. This method synthesizes individual network flows and demonstrates superior performance compared to the current state-of-the-art GAN-based methods. The proposed approach provides a more accurate and efficient way of generating realistic traffic, addressing the limitations of existing models. It is explained in Section 4.3 and evaluated using our benchmark (see below) in Section 4.6.

- **Evaluation Benchmark for Synthetic Network Flows:** The third contribution is the development of a comprehensive evaluation benchmark designed to assess the quality of synthetic network flows. This benchmark covers multiple dimensions of traffic generation, including realism, diversity, novelty, and compliance, providing standardized metrics that can be used to compare various generative models. It serves as a critical tool for the community, enabling consistent and meaningful comparisons between different approaches. It is presented in Section 4.4.

- **Pattern Mining-based Network Traffic Generation:** The fourth and final contribution is the development of a generation method based on pattern mining techniques, which builds upon the Bayesian Network-based approach introduced in the third contribution. This method produces high-quality synthetic network flows with greater diversity and fidelity, making it a potential first substitute for traditional simulations. By leveraging recurring patterns in network behavior, this approach offers a new direction for network traffic generation, opening up possibilities for more realistic and adaptable synthetic datasets. It is presented in Section 5.4 and evaluated using our benchmark (see above) in Section 5.6.

1.5 Plan of the Study

We begin by introducing foundational concepts to ensure that readers unfamiliar with the field can follow the research presented. This foundational section covers the theoretical background on network traffic, intrusion detection systems, and the role of synthetic data generation, providing essential context for the following discussions.

Next, we review the state of the art in synthetic network traffic generation. This overview covers various generative models proposed for this task, such as GAN, Variational Autoencoders, and Diffusion Models. We also outline the different criteria used to evaluate generated traffic quality, highlighting the strengths and limitations of existing approaches.

Following this, we introduce a novel method for generating individual network flows based on Bayesian Networks. Evaluated using a benchmark developed specifically for this research, this method is compared against state-of-the-art GAN-based models to assess the quality of the generated flows with respect to a source dataset. We emphasize the advantages of using Bayesian Networks for network flow generation, presenting evidence that this approach yields more accurate and scalable results.

In contrast to the Bayesian Network-based method, the final contribution presents an innovative method for generating network traffic based on Pattern Mining, which not only produces realistic and diverse traffic but also preserves temporal dependencies between flows. This ability to capture sequential relationships among flows makes it particularly valuable for scenarios where these temporal dynamics are essential. We also discuss how this method can potentially replace traditional network traffic simulations, especially in scenarios where scalability and adaptability are crucial.

Chapter 2

Background

This chapter provides a comprehensive foundation for understanding the fundamental concepts relevant to the research presented in this thesis.

This chapter begins with an overview of network basics, including the fundamental principles of data exchange over computer networks, the role of users and network packets, and the various methods for analyzing packet exchanges. The chapter then delves into machine learning, outlining its significance and workflow. Furthermore, it explores advanced unsupervised learning techniques for data generation, such as Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), and Bayesian Networks (BN), highlighting their training processes and challenges. Lastly, the chapter addresses common pitfalls in tabular data generation.

2.1 Network Basics

Network traffic refers to the data exchanged over a computer network, encompassing all digital communications between connected devices. To effectively analyze network traffic, it is essential to understand how data flows across various layers of a network. The structure and behavior of this flow can be explained using the Open Systems Interconnection (OSI) model, which provides a framework for conceptualizing network communication. The OSI model divides network communication into seven distinct layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application [24]. This layered approach allows for a clearer understanding of where and how data is exchanged between devices.

Before delving into traffic analysis, it is crucial to define two key concepts: what constitutes a user and a packet. After introducing these concepts, we will explain how packets are exchanged in a network and how their exchange can be analyzed to study network traffic.

2.1.1 Users on a Network

A user on a network is any entity (person, device, or software) that sends or receives data across that network. In the context of network communication, particularly at the Internet layer

(Layer 3) and the Transport layer (Layer 4) of the TCP/IP model, these users are identified by their IP addresses, which are unique to each device on the network.

There are two main types of IP addresses: IPv4 and IPv6. IPv4 uses a 32-bit address scheme, allowing for around 4.3 billion unique addresses [25], while IPv6, developed to address the exhaustion of IPv4, uses a 128-bit address scheme, offering a much larger address space [26], [27]. However, this thesis focuses solely on IPv4 addresses, as they remain prevalent in many existing networks. Unless mentioned otherwise, an IP address in this thesis refers to an IPv4 address.

IP addresses can be classified into public and private addresses. Public IP addresses are unique across the entire internet, allowing devices to communicate globally. Private IP addresses, such as 192.168.0.1, are used within local networks and are not routable on the public internet. These private addresses are often used in conjunction with Network Address Translation (NAT) to allow multiple devices on a local network to share a single public IP address. It's also important to note that a single network interface with a unique MAC address can support multiple IP addresses, allowing a device to participate in multiple networks simultaneously.

2.1.1.1 Application Ports

At the Transport layer (Layer 4), port numbers can further distinguish users. Ports identify specific processes or services on a device, enabling multiple applications to use the network simultaneously. For example, web traffic typically uses port 80 for HTTP or port 443 for HTTPS, while email traffic might use port 25 for SMTP. This distinction allows the same IP address to support different types of communication for various applications on the same device.

In addition to such well-known ports, there are ephemeral ports, which are temporary port numbers used by clients for short-lived connections. When a client initiates a connection (such as opening a web page or sending an email), it is assigned an ephemeral port. This allows the client to uniquely identify its connection to the server, after which the port is relinquished once the communication ends [28].

2.1.2 Network Packets

Network traffic consists of packets, which are structured units of data containing both the transmitted content and essential metadata, such as sender, recipient, and the application or service in use. The book by Tanenbaum et al. [29] provides a detailed breakdown of these packets across different network layers:

- **Layer 3 (Internet Layer):** This layer is responsible for packet forwarding, including routing through different routers. The data unit at this layer is called a *packet*. The header of a Layer 3 packet includes fields such as the source and destination IP addresses, which are crucial for routing the packet across networks.

- **Layer 4 (Transport Layer):** This layer ensures that data is transferred from one point to another reliably and in the correct order. The data unit at this layer is called a *segment* (in TCP) or a *datagram* (in UDP). The header at this layer includes information such as port numbers, which identify the specific process or service that the data is associated with, as well as sequence numbers in TCP to ensure that segments are reassembled in the correct order.

- **Layer 5 (Application Layer):** This is where the actual user data resides. It includes the application-specific data to be transmitted, such as a piece of an email, a file segment, or part of a web page. When a packet reaches this layer on the receiving end, the data is extracted and processed by the appropriate application.

In summary, a network packet typically consists of a **header** and a **payload**. The header contains metadata required by the lower layers for routing, delivery, and reassembly, while the payload carries the actual data being transmitted. This structure, aligned with the encapsulation process in the TCP/IP model, is illustrated in Figure 2.1.

This concept of encapsulation is crucial to understanding how data is transmitted across networks, as it allows different layers of the network stack to operate independently and flexibly. [29]

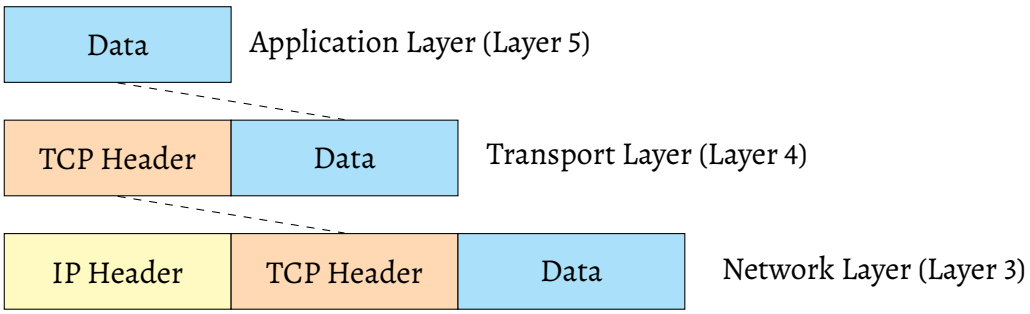

Figure 2.1: Encapsulation concept. The IP Header and TCP Header are called the *header* of the packet, while the Data are called the *payload*

2.1.3 Analyzing Packet Exchanges

The exchange of packets can be analyzed to diagnose network issues, identify security threats, and understand the nature of the traffic [29]. This analysis involves various techniques focusing on different objects of interest, such as packets, flows, and specific network features.

Each technique provides unique insights and helps understand user activities on a network. Below, we introduce the primary methods of analysis: packet inspection, network flow analysis, and feature analysis.

2.1.3.1 Packet Inspection

Packet inspection involves examining individual packets to extract information from their headers and payloads. Common tools for packet inspection include Wireshark[1] and tcpdump[2]. Packet inspection can reveal detailed information about protocols, communication patterns, and potential anomalies [30].

2.1.3.2 Network Flow Analysis

A network flow is a sequence of packets sent from a source to a destination that share common attributes such as IP source, IP destination, Source Port, Destination Port, and protocol. Flows can be unidirectional or bidirectional. A unidirectional flow is a sequence of packets flowing in one direction only, while a bidirectional flow includes packets flowing in both directions between a source and destination [31].

Figure 2.2 shows the packet exchange between two hosts and the different flow groupings based on whether it is unidirectional or bidirectional. By grouping and analyzing packets at the flow level, one can provide a higher-level view of network activity than individual packet inspection [31]. Standard tools for network flow analysis include nfdump [32], CICFlowmeter [7], and nProbe [33].

2.1.3.3 Feature Analysis

Feature analysis involves examining specific characteristics of network traffic. These features include throughput (the rate of successful data transfer), latency (the time taken for data to travel from source to destination), and jitter (the variation in packet arrival times). Such features help detect network misuse and security breaches and optimize traffic management [30].

[1] https://www.wireshark.org/
[2] https://www.tcpdump.org/

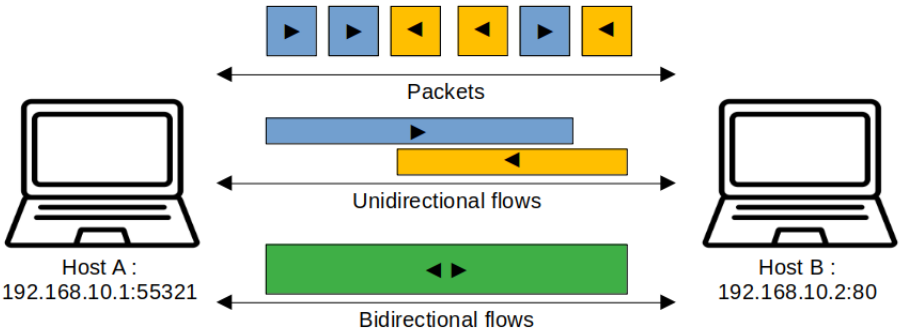

Figure 2.2: Exchange of packets between two hosts, with the direction of each packet indicated by the arrow inside it. A unidirectional flow groups the packets in the same direction, and a bidirectional flow group groups packets regardless of direction. The host is identified by their IP and Port in the format IP: Source Port

2.1.4 Network Flow Format

Network flow analysis stands out among the different analysis techniques presented above, especially in the context of NIDS. Unlike packet inspection, which can be overly detailed and cumbersome, or feature analysis, which might overlook critical interactions between packets [34], network flow analysis provides the right level of granularity to effectively monitor and analyze network traffic [35]–[37].

As said previously, packets are grouped by a set of common attributes in network flow analysis. A typical example is the five-tuple: IP source, IP destination, Source Port, Destination Port, and Protocol (this is the set of characteristics used to group packets in IPFIX [38] and Biflows [7]). Other grouping strategies can exist, such as aggregating the packets per seven key characteristics (NetFlow v9): IP source, IP destination, Source Port, Destination Port, Protocol, Type of Service (ToS), and Input Interface. However, these grouping strategies can be adapted depending on the use. In the rest of the thesis, we only consider one level of granularity and refer to network flow as grouping by the five characteristics.

The grouping of packets based on shared characteristics results in a new network flow object, which aggregates information from the packet level, such as the total number of bytes across all packets, the time difference between the emission of the first packet and the reception of the last packet, and other network attributes. Therefore, a network flow can be defined by the combination of (IP source, IP destination, Source Port, Destination Port, Protocol) and a set of additional aggregated features. The specific features considered in describing a flow can vary, leading to different flow formats [38]–[42]. Table 2.1 shows the various feature sets of some of the most popular network flow analysis protocols with their descriptions. It shows

that all the different network flow tools define a flow by at least the 5-tuple (IP source, IP destination, Source Port, Destination Port, Protocol). Some also include the ToS and the input interface. However, most of these tools are, however, configurable, allowing for a modification of the set on which the aggregation is made or the set of features extracted.

Feature	NetFlowv9 [39]	IPFIX [38]	Biflows [7]	J-Flow [41]	NetStream [42]
Source IP Address	✓	✓	✓	✓	✓
Destination IP Address	✓	✓	✓	✓	✓
Source Port	✓	✓	✓	✓	✓
Destination Port	✓	✓	✓	✓	✓
Protocol	✓	✓	✓	✓	✓
Flow Start Time	✓	✓	✓	✓	✓
Flow End Time	✓	✓		✓	✓
Packet Count	✓	✓	✓	✓	✓
Byte Count	✓	✓	✓	✓	✓
Flow Duration	✓	✓	✓	✓	✓
Next Hop IP Address	✓	✓		✓	✓
Src Autonomous System Number	✓	✓		✓	✓
Dst Autonomous System Number	✓	✓		✓	✓
Source MAC Address	✓	✓		✓	✓
Destination MAC Address	✓	✓			✓
VLAN ID	✓	✓		✓	✓
Type of Service (ToS)	✓	✓		✓	✓
TCP Flags	✓	✓	✓	✓	✓
Input Interface	✓			✓	✓
Output Interface	✓	✓		✓	✓
Application ID		✓			
URL and HTTP Host Information		✓			
Flow Sampling Interval		✓			
BGP Next Hop IP Address		✓			
Packet Inter Arrival time			✓		
Header length			✓		

Table 2.1: Comparison of Network Flow Features across different formats. A ✓ indicates that, in this format, this feature is used for defining the scope of the flow

2.1.4.1 Similarity with Tabular Data

This network flow abstraction allows us to treat a set of recorded network flows as tabular data. In this framework, each network flow corresponds to a row in a table, and the network features correspond to the columns. This similarity, noted in various studies [11], [19], [43]–[47], enables the use of tabular data analysis tools for network flow analysis, leveraging established techniques for more effective analysis. Regarding network flow generation, we can mainly leverage tabular data generation tools to generate synthetic network flows.

2.2 Introduction to Machine Learning

Machine learning is a field of artificial intelligence that enables computers to learn from data and make decisions. It involves training a model using specific algorithms to identify patterns and relationships within data [48]–[50]. Machine learning applications span diverse fields such as healthcare and finance, where they improve decision-making and automate complex tasks [49], [51].

2.2.1 Machine Learning Pipeline

A systematic pipeline is typically followed to build effective machine learning models, including critical steps like data collection, model class identification, data preparation, model training, and model evaluation [52], [53]. This pipeline ensures that models are created efficiently and can generalize well to unseen data.

- **Data Collection**: Gathering relevant, high-quality data is foundational for learning patterns accurately. This step often involves extracting data from databases, online sources, or sensors and ensuring it is representative of the problem domain [53].

- **Model Class Identification**: In this step, we identify a class of models (such as neural networks, see below) that fits the task's requirements for accuracy, interpretability, and computational efficiency [52]. This initial choice helps narrow down models suited for the learning problem.

- **Data Preparation**: Preparing data to be model-ready involves cleaning and structuring it by handling missing values, normalizing ranges, and organizing it into a suitable format [52]. Well-prepared data enhances model performance.

- **Model Training**: The model learns from the data in this step by adjusting parameters through methods like gradient descent, a process critical for learning meaningful patterns and relationships [54].

- **Model Evaluation**: Testing trained models on new data helps verify their predictive ability and determine how well they generalize to new, unseen scenarios [52].

2.2.2 Neural Networks

Based on the insights presented in the book by Yann LeCun et al. [55], neural networks are a foundational framework in machine learning, particularly for tasks involving complex data patterns. They consist of interconnected layers of nodes (neurons) that process input data and learn to make predictions or decisions.

A neural network typically consists of an input layer, one or more hidden layers, and an output layer. Each neuron receives inputs, processes them using weights and biases, and passes the output through an activation function. Specifically, each input is multiplied by a corresponding weight, and then a bias is added to this weighted sum. The activation function is then applied to this result. An activation function is a mathematical function that determines whether a neuron should be activated or not. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns. Common activation functions include the sigmoid function, which outputs values between 0 and 1, the hyperbolic tangent (tanh) function, which outputs values between -1 and 1, and the rectified linear unit (ReLU), which outputs the input directly if it is positive; otherwise, it outputs zero.

Training neural networks involves optimizing the weights and biases to minimize the error between the predicted and actual outputs. This process uses algorithms like stochastic gradient descent, which iteratively adjusts the parameters to reduce the error.

2.3 Unsupervised Learning for Data Generation

Unsupervised learning is a class of machine learning techniques that relies on unlabeled data, aiming to uncover underlying structures or patterns within the dataset without any predefined target outcomes [56]. Data generation is a form of unsupervised learning that consists of creating synthetic datasets. This approach is essential in fields where privacy concerns or data scarcity limit access to real datasets, as it enables the creation of data that closely mimics real-world characteristics. In this work, we focus on three widely used methods for synthetic data generation: Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), and Bayesian Networks.

2.3.1 Generative Adversarial Networks (GAN)

GAN consist of two neural networks: the generator and the discriminator. The generator takes random noise as input and generates synthetic data, while the discriminator evaluates whether

the input data is real or fake [13].

2.3.1.1 Training Process

The training process of GAN involves an adversarial process where the generator and discriminator are trained simultaneously. The generator tries to fool the discriminator by producing realistic data, while the discriminator tries to distinguish between real and fake data correctly. This competition drives both networks to improve. Figure 2.3 illustrates the training process of a GAN.

2.3.1.2 Generation Process

After being trained, the generator can turn a random vector into new samples indistinguishable from proper original data. This is the operation that allows GAN to generate new data samples.

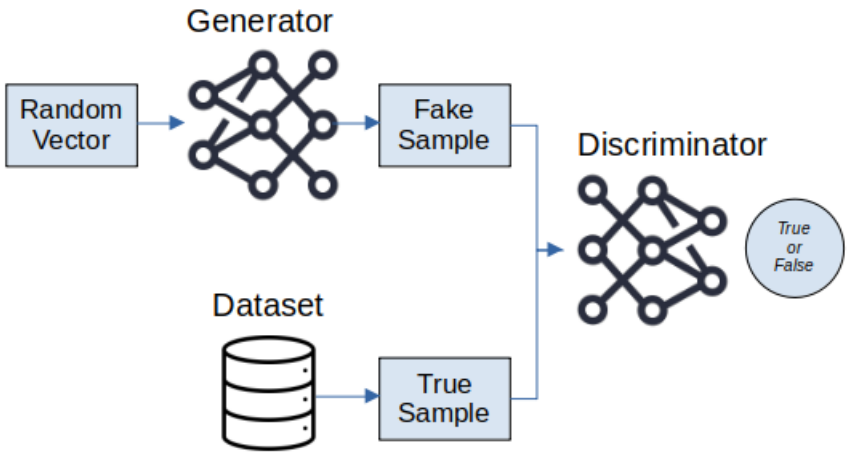

Figure 2.3: Training process of a GAN. The generator creates data samples from random noise, which are then evaluated by the discriminator against real data samples. The discriminator's feedback helps improve both the generator and itself.

2.3.1.3 Challenges

Training GAN can be challenging due to the delicate balance between the generator and discriminator, often resulting in training instability [13], [57]. This process requires careful tuning of hyperparameters and network architectures to achieve stable training. Mode collapse is an-

other common challenge, where the generator produces a limited variety of outputs and fails to capture the diversity of the real data [58].

2.3.2 Variational Autoencoders (VAE)

Variational Autoencoders (VAE) consist of two neural networks: the encoder and the decoder. The encoder transforms input data into a probability distribution in a lower-dimensional latent space, and the decoder generates synthetic data from this distribution [59].

2.3.2.1 Training Process

The training process of VAE involves a variational approach, encouraging the latent space to follow a known distribution, typically a standard Normal distribution. The encoder learns to produce the parameters μ (mean) and σ (standard deviation) of this distribution for any given input. The loss function for the encoder and decoder includes the reconstruction loss (measuring how well the encoder/decoder structure reconstructs a given input) and the KL divergence (calculating the difference between the learned latent distribution and a predefined prior distribution, usually a standard Normal distribution). Figure 2.4 illustrates the training of a VAE.

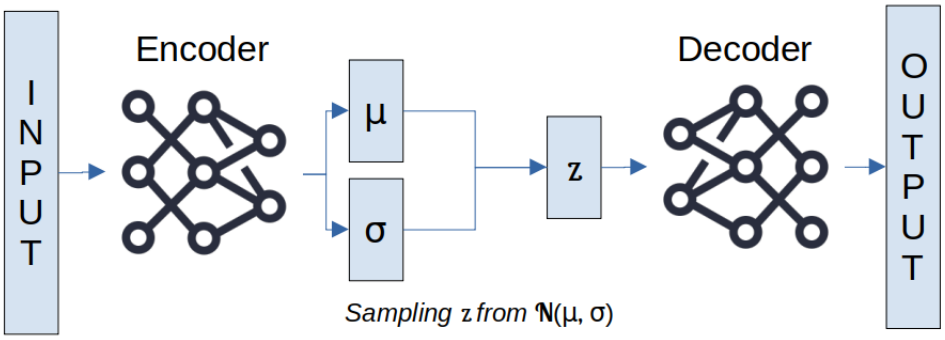

Figure 2.4: The training process of a VAE. The encoder learns to represent the training samples in a latent distribution characterized by a mean μ and a standard deviation σ. The learned latent distribution is encouraged to be close to a standard Normal distribution (the prior distribution). From this learned distribution, we can sample a latent variable z, which the decoder uses to generate a reconstruction of the input.

2.3.2.2 Generation Process

After training, the generation process in a VAE involves sampling a latent variable z from the reference prior distribution. This reference prior distribution is typically a standard Normal

distribution, which is a Gaussian distribution with a mean of 0 and a variance of 1 (like in Figure 2.4). In simpler terms, we generate random values for z that follow this standard Gaussian distribution. The decoder then transforms this z back into the data space, generating new synthetic samples that resemble the original training data.

2.3.2.3 Challenges

Training VAE presents challenges, such as balancing reconstruction and regularization. The trade-off between reconstruction loss and KL divergence can be challenging to manage, requiring careful tuning [60]. Posterior collapse, where the decoder ignores the latent space and relies solely on its internal weights, is another issue, leading to less meaningful latent representations [61].

2.3.3 Bayesian Networks for Data Generation

Unlike GAN and VAE, Bayesian Networks do not rely on deep learning. A Bayesian Network is a statistical model that explicitly encodes the conditional probabilities between variables within a learning problem. It is presented as a probabilistic graphical model, where nodes represent each random variable in the problem, and the links between them denote the conditional probabilistic dependencies between pairs of variables [62].

2.3.3.1 Training Process

Training Bayesian Networks involves two main steps: structure learning and parameter learning.

Structure learning involves determining the network's structure, which can be done using various algorithms that fall into two broad categories: constraint-based methods and score-based methods. Constraint-based methods use statistical tests to identify dependencies and independencies between variables, constructing the network structure by ensuring that the resulting network satisfies these constraints. On the other hand, score-based methods evaluate different possible structures based on a scoring function (such as the Bayesian Information Criterion or BIC[63]) and select the structure that best fits the data according to this score.

Parameter learning calculates the conditional probability distributions for each node, given its parents in the network. This involves estimating the probabilities that define how each node depends on its parents. This typically means calculating conditional probability tables (CPTs) for discrete variables. For continuous variables, parameter learning might involve estimating

parameters for a specific distribution, such as the mean and variance for a Gaussian distribution. Figure 2.5 illustrates a Bayesian Network.

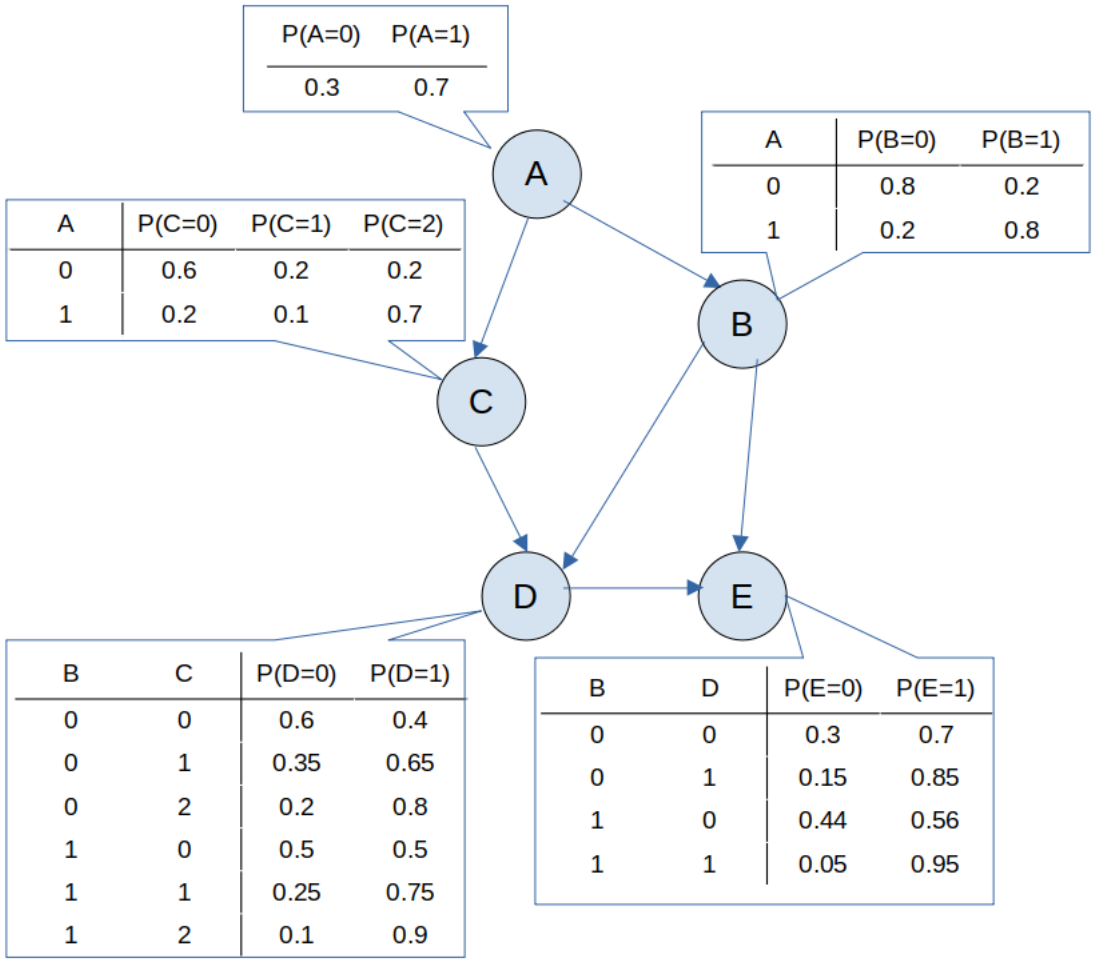

Figure 2.5: Bayesian Network representing the dependencies between five discrete variables A, B, C, D, and E. The arrows indicate dependency relationships, while nodes are associated with conditional probability tables.

2.3.3.2 Generation Process

To generate data using a Bayesian Network, we sample from the conditional probability distributions defined by the network structure. The process begins with the root node(s) (A in Figure 2.5), where we sample values directly from their marginal distributions because they have no parents. For each subsequent node, we sample values based on the conditional distributions given their parent nodes' sampled value using the Bayes rule.

For discrete variables, the conditional probability distributions are typically represented by conditional probability tables (CPTs). Each entry in a CPT specifies the probability of the discrete variable taking on a particular value given the values of its parent nodes. This process continues until values have been sampled for all nodes in the network, resulting in a complete synthetic data instance.

For continuous variables, the conditional distributions might be represented by specific parametric forms, such as Gaussian distributions. In this case, the distribution parameters (e.g., mean and variance) are functions of the parent nodes' values. Sampling for continuous variables involves generating values from these conditional distributions using the estimated parameters.

2.3.3.3 Challenges

Challenges in training Bayesian Networks include scalability, complex dependencies, and data sparsity. Large numbers of variables can make the computations expensive, and accurately capturing dependencies requires careful consideration of the network structure [64].

2.4 Challenges in Generating Synthetic Tabular Data

As mentioned in Subsection 2.1.4, network flows can be viewed as a specific type of tabular data. Consequently, generating synthetic network flows is essentially an application of tabular data generation. Therefore, it is pertinent to highlight some of the challenges associated with generating synthetic tabular data in this context.

2.4.0.1 Mix of Categorical and Numerical Data

Tabular records often contain mixed data types (discrete and continuous). GAN and VAE, designed for continuous data, struggle with discrete data due to gradient descent issues [65]. While Bayesian Networks can theoretically handle both discrete and continuous variables, practical implementations in Python often fall short. Libraries like pgmpy[3] and bnlearn[4] have limited support for hybrid models. As a result, fully leveraging Bayesian Networks for mixed data types often requires manual intervention or specialized software not readily available in popular Python libraries.

[3] https://pgmpy.org/
[4] https://erdogant.github.io/bnlearn/

2.4.0.2 Sparse Distribution

Tabular data often exhibit sparse distributions, especially when dealing with high-cardinality categorical variables. When using one-hot encoding—a process that converts categorical variables into a binary vector with a single high (1) value corresponding to the category and the rest low (0)—for discrete features in neural networks, this results in high-dimensional and sparse input spaces. This can lead to issues like mode collapse in GAN or posterior collapse in VAE [57], [66]. For Bayesian Networks, sparse distributions and high-cardinality features pose challenges due to the complexity of CPTs. As the number of categories increases, the CPTs become large and sparse, making the computations more complex [67].

2.4.0.3 Dependencies Between Features

Tabular data often have complex dependencies between variables that need accurate modeling. For example, in network data, the Source Ports and Destination ports depend on each other and rely on the Transport Protocol. GAN and VAE have limited capacity to capture these dependencies [65], while Bayesian Networks are better suited for handling interdependent data structures [67].

2.5 Summary

This chapter provides a comprehensive foundation for understanding the fundamental concepts relevant to the research presented in this thesis. We begin with an overview of network traffic basics, detailing the fundamental principles of data exchange over computer networks, the role of users, and the nature of network packets. We then delve into the realm of machine learning, highlighting its significance and workflow.

Further, we explore advanced unsupervised learning techniques for data generation, such as Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), and Bayesian Networks (BN). We discuss their training processes, the generation of synthetic data, and the specific challenges each method faces, particularly when handling tabular data. GAN and VAE, designed primarily for continuous data, encounter difficulties with discrete data. In contrast, Bayesian Networks, although capable of handling both data types, often struggle with practical implementation in commonly used Python libraries.

Lastly, we address common pitfalls in data evaluation methods for tabular data generation, including issues arising from mixed data types, sparse distributions, and complex dependen-

cies between features. These challenges highlight the need for robust methodologies and tools to generate and evaluate synthetic data effectively.

The insights gained from this chapter set the stage for the subsequent analysis and discussions in this thesis, providing the necessary background to understand and address the complexities of synthetic data generation and network traffic analysis.

Chapter 3

State of the art

Obtaining real network traffic for cybersecurity evaluation is impractical due to privacy concerns and the difficulty of labeling data accurately, making it unsuitable for testing network intrusion detection systems (NIDS) [68]. While simulations offer a solution by generating labeled traffic in controlled environments, they are limited by their static, predefined configurations, which restrict their adaptability and increase their cost as the number of hosts or activities grows [9], [69].

Synthetic traffic generation offers a potential solution by replicating real-world network behaviors without leaking sensitive data or requiring manual labeling. These techniques provide valuable scalability and adaptability. However, their ability to fully overcome the challenges posed by real and simulated traffic remains largely hypothetical and is an area of ongoing research.

As already mentioned in Section 1.3, we reduce the scope of our study to the generation of synthetic benign network flows. Indeed, our goal is to generate traffic for evaluating NIDS, and most NIDS are operating at the network flow level [70]. Similarly, we are only concerned about the availability of benign traffic, leaving aside the generation of malicious events.

Therefore, this state-of-the-art review explores the key AI models and methodologies employed for generating synthetic benign network flow and evaluates such generation. In Section 3.1, we first explain how current research manages the generation of network flow by AI methods, and then, in Section 3.2, we endeavor to define what constitutes a good network flow generation and how to assess its quality.

3.1 Synthetic network flow generation using AI

The generation of synthetic network flows has emerged as a crucial area of research for improving the performance of NIDS [9]. By replicating the characteristics of real network traffic, synthetic flows provide diverse and comprehensive datasets that help train NIDS.

Artificial intelligence (AI) plays a pivotal role in this process by enabling the creation of syn-

thetic flows that closely mirror real-world traffic patterns. Generative models learn from network data and can replicate intricate patterns and generate realistic traffic scenarios. Techniques like Generative Adversarial Networks (GAN) are mainly used for this task, as they generate data that mimics actual network behavior. These AI-driven approaches not only enhance the diversity of training datasets but also improve the accuracy of NIDS by simulating a wide range of traffic conditions.

This section presents two main applications of AI-driven synthetic flow generation: the first focuses on producing flows specifically tailored to improve a specific NIDS, ensuring the system is exposed to diverse benign behaviors. The second involves generating general-purpose synthetic benign network flows that can be used for broader research purposes without being tied to any specific NIDS.

3.1.1 Generating Network Flows in the Context of Training an NIDS

In this Subsection, we explore approaches that focus on generating synthetic network flows to enhance the training of an NIDS. These methods aim to augment the training dataset with synthetic flows that resemble real traffic, thereby improving the NIDS's ability to detect anomalies. This enhancement is particularly critical as it allows the NIDS to cover the patterns of regular traffic better and differentiate them from malicious activities. The models discussed here leverage generative techniques to create diverse and realistic network flows, which are then used to train the NIDS, ensuring it is exposed to a wide range of scenarios during training.

Zenati et al. [71] make the first attempt to enhance NIDS performance by generating synthetic network flows. They propose a GAN-based method that creates synthetic network flows to expand the training dataset for an NIDS. In their approach, the NIDS is an autoencoder trained to reconstruct regular network flows. After training, the autoencoder flags poorly reconstructed network flows as potential anomalies, assuming they are unseen during training and thus abnormal. The goal is to generate synthetic network flows that closely resemble the input flows, thereby improving the NIDS's training process. To achieve this, the authors utilize a BiGAN structure. In BiGAN, two discriminators are employed: one to distinguish between real and synthetic network flows, and the other to differentiate the latent representations of these flows produced by the NIDS's encoder [72]. This process is illustrated in Figure 3.1. This dual-discriminator setup helps stabilize the training process and ensures that the generated flows effectively enhance NIDS performance.

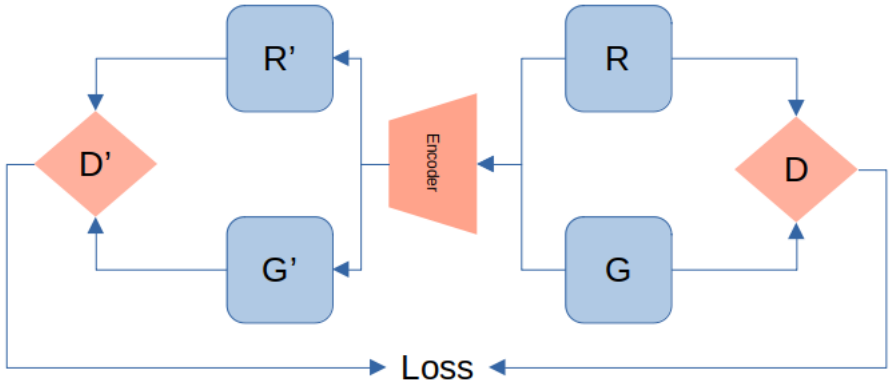

Figure 3.1: BiGan structure used in ALAD. R are the real network flows, and G is the network flows generated by the generator. The encoder is the encoder part of the NIDS; it is used to get the projection of both R and G in its latent space, and two discriminators, D and D', are trained to differentiate both R from G and R' from G'. The loss of the two discriminators becomes the loss of G.

Zenati et al.'s method, ALAD, is tested on the KDD99 dataset[1] by evaluating the precision, recall, and F1-score of their NIDS. Their auto-encoder NIDS outperforms the state-of-the-art ML anomaly detection method, DAGMM. Their BiGAN training approach is also adopted by Xu et al. [73] using the CIC-DDoS2019 dataset[2], and by Zixu et al. [74] with the UNSW Bot-IoT dataset[3]. Both studies demonstrate substantial performance improvements when comparing training without generated data to training with generated data. In both papers, the focus of the generation is on the performance gain in NIDS and not the quality of the generated traffic. Therefore, no evaluation is conducted on the generated traffic.

Han et al. [75] also explore using GAN to train a Gated Recurrent Unit with an attention mechanism for traffic classification. They employ a Wasserstein GAN [76] to generate synthetic flows, which were used in training their classifier. Their method is tested on the ISCX-2012[4] and ISCX-2017[5] datasets, demonstrating enhanced classifier performance with the inclusion of generated traffic during training, but here again, there is no evaluation of the quality of the traffic that is used to improve the classifier.

The application of generative methods in the studies discussed is inherently limited by their focus on enhancing a single specific NIDS. These methods are designed and optimized to im-

[1] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[2] https://www.unb.ca/cic/datasets/ddos-2019.html
[3] https://research.unsw.edu.au/projects/bot-iot-dataset
[4] https://www.unb.ca/cic/datasets/ids.html
[5] https://www.unb.ca/cic/datasets/dos-dataset.html

prove the performance of the NIDS for which they are developed, often leading to tight integration between the generative models and the specific characteristics of the NIDS. Consequently, it can be inferred that these approaches may not generalize well beyond their intended context, making them less suitable for broader purposes, such as general network traffic generation. To our knowledge, no existing studies provide a framework or methodology that enables these generative models to produce synthetic network flows independently of specific NIDS.

3.1.2 Generating Network Flows for General Purposes

In the previous Subsection, we highlighted the limitations of application-specific generative methods designed for enhancing NIDS. In contrast, this Subsection focuses on generating network flows without assuming any particular deployment context for the synthetic traffic. These research projects aim to replicate the characteristics of the training network closely, targeting the broader applicability of synthetic traffic generation (like IoT network optimization [77]). Two main challenges have emerged in the state of the art: generating network flow features and including temporal dependencies in the generated network flows.

3.1.2.1 Individual Network flow Generation

Ring et al. [16] are the first to tackle the individual generation of network flows using WGAN-GP. We refer to their process as individual generation because it focuses on creating network flows independently, without considering the temporal relationships between them. They model network flows as tabular data, reducing the problem of generating new synthetic rows in a table without considering previous rows. As discussed in Section 2.4, handling mixed data types is one challenge in using GAN for tabular data. They, therefore, convert categorical features into continuous representations using IP2Vec embeddings [78] to address this issue.

An embedding is a representation of a discrete value in a continuous vector space. An IP2Vec embedding specifically encodes the discrete attributes of a network flow, such as IP addresses, application ports, or transport protocols, into continuous representations. The continuous representation for a given discrete value is based on patterns and relationships with other discrete values in the dataset, which are learned by a simple Dense layer of a neural network. To do so, the embedding model needs to be trained to predict a discrete value based on the discrete values of its context (see Figure 3.3). Once learned, the embedding model enables the encoding of a discrete value into a continuous representation by taking the weights of the dense layer that are activated by the discrete value as an embedding vector for that discrete value (see

Figure 3.2). The mixing of data types during GAN training is thus avoided.

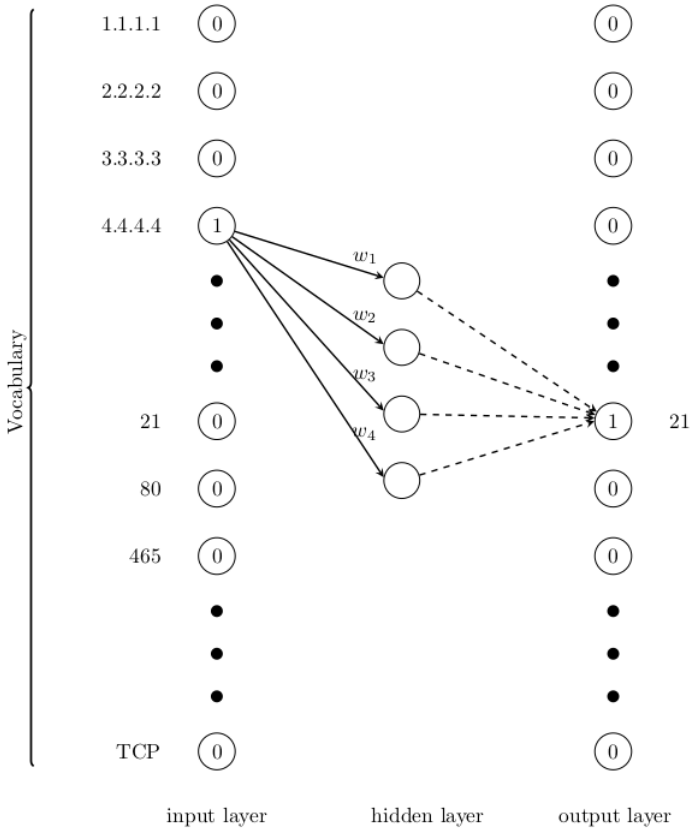

Figure 3.2: Architecture of the neural network used by IP2Vec. Image taken from [78]. For clarity, only the weights of one input and one output neuron are drawn. IP2Vec comprises a dense layer between the input and the output layer of the vocabulary size. In this example, the neural network is trained with the sample (4.4.4.4, 21). During the training, the one-hot encoding of the port '21' value should be predicted based on the input of the one-hot encoding of '4.4.4.4'. After the training step, the vector with the components $w1, w2, w3$ and $w4$ (the weights activated by the input '4.4.4.4') is the continuous representation of the IP Address 4.4.4.4.

Ring et al. trained their WGAN-GP model using the Two Time-Scale Update Rule (TTUR) [79], where the discriminator and generator are updated with different learning rates. This method helps achieve more stable and effective training. The approach was evaluated on the CIDDS-001 dataset[6] by examining protocol compliance and attribute distributions. The generated traffic had attribute distributions similar to the training traffic, particularly when using the IP2Vec embeddings, compared to representing discrete values by binary or numeri-

[6] https://www.hs-coburg.de/forschung/forschungsprojekte-oeffentlich/informationstechnologie/cidds-coburg-intrusion-detection-data-sets.html

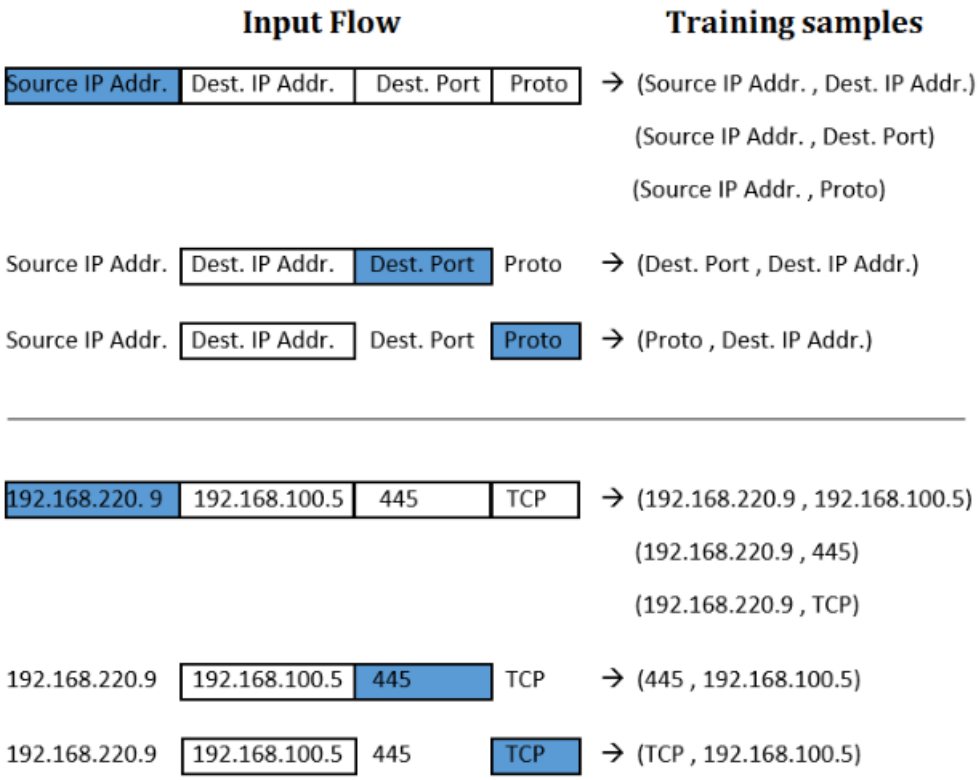

Figure 3.3: Generation of training samples for IP2Vec. Image taken from [78]. The upper part of the figure shows the general process of training sample generation. *Input words* are highlighted in blue, and *context words* are highlighted in black frames with a white background. The right side of the figure shows the generated training samples for the corresponding combinations of *input word* and *context words*. The lower part of the figure provides an example and shows the generated training samples. The objective of the training is for the embedding model to predict the *Input word* based on each *context word*

cal representations.

Manocchio et al. [80] address the other major issue of GAN for tabular data mentioned in Section 2.3: mode collapse. They adapt a Wasserstein GAN [76] to mimic traffic from an Australian ISP backbone dataset, implementing a BiGAN like Zenati et al. [71]. Their model, FlowGAN, features two discriminators: one discriminating the network flows generated by the generator and another discriminating their latent representations given by a pre-trained encoder. They utilize IP2Vec embeddings to encode discrete features. This work can be seen as an advancement from the work of Ring et al. [16], specifically focusing on reducing mode collapse.

The generated flows are evaluated based on their duration and packet size distributions, demonstrating reduced mode collapse. However, one issue not addressed is the learning of dependencies among features. As discussed in Section 2.4, GAN struggle with learning dependencies in tabular data. In the results shown in Table 3.1, there appear to be some unrealistic correlations in the generated data. For instance, some generated flows exhibit ephemeral ports (see Section 2.1) as both source and destination, which is highly unlikely in real datasets. This anomaly could be specific to the dataset used; however, verifying this would require access to the original data, which is unfortunately not possible since they used a private Australian ISP dataset. The question of whether or not their model learns the correlation between the different features correctly remains open.

Bourou et al. [46] specifically target this question of learning correlations across features. In their paper, they explore the possibility of adapting GAN that are initially designed for general tabular data, to the domain of network flow data. They specifically investigate the implementation of three GAN models: TableGAN [81], CTGAN, and CopulaGAN (those last two are presented in the reference [82]).

TableGAN utilizes a generator and discriminator based on a CNN architecture. It also includes an additional discriminator network, termed the *classifier* C, which is trained on the real data to predict the value of one feature based on the values of others. By training a separate classifier for each feature, this setup enforces that the generated data respects the dependency relationships present in the training data.

CTGAN focuses more on avoiding mode collapse. It achieves this using a Variational Gaussian Mixture Model to normalize continuous features rather than a simple min-max normalization. To further reduce mode collapse in categorical features, the discriminator is designed to consider all categories of a given feature. The training process is modified to ensure that all categories in discrete columns occur evenly, making the generator a conditional generator.

CopulaGAN is a variation of CTGAN that emphasizes learning the dependencies among

variables. Copulas are utilized to describe the intercorrelations between features. During training, CopulaGAN not only learns the probability distributions of each table column but also the correlations between these columns. These three models are compared on the task of reproducing the NSL-KDD dataset[7], assessing their effectiveness in capturing the complex dependencies in the data.

The three studies mentioned above focus exclusively on using GAN, even though, as highlighted by Bourou et al. [46], the specific task of network flow generation closely resembles tabular data generation. However, other methods for generating tabular data exist. For instance, as discussed in the background, Bayesian Networks have already demonstrated superior performance compared to GAN in specific contexts, such as medical patient data synthesis [83]. Consequently, there is a noticeable gap in the literature regarding a comprehensive comparison between GAN and Bayesian Networks for network flow generation.

Src. IP	Dest. IP	Src. Port	Dest. Port	In Packets	Out Packets	In Bytes	Out Bytes	Duration
10.0.0.29	10.0.2.137	161	1039	4	0	428	0	21
10.0.1.22	10.0.0.85	53213	53213	1	0	237	0	0
10.0.0.11	172.20.10.23	88	52076	5	5	2027	320	20
10.0.2.203	172.20.10.25	443	51905	1	1	93	40	1
10.0.0.11	172.20.10.23	88	58571	7	6	3415	2038	3
172.20.10.24	10.0.0.202	59898	53213	1	2	237	474	1
10.0.1.27	10.0.0.150	55798	53213	1	0	237	0	0
10.0.1.27	10.0.0.191	53210	53213	2	0	474	0	0
10.0.1.27	10.0.0.122	55798	53213	1	0	237	0	0
10.0.0.231	10.0.5.134	53213	53213	1	0	237	0	0

Table 3.1: Network flow generated by FlowGAN. The table is directly extracted from [80]. On the last five flows, both the Source Port and the Destination Port are within the ephemeral port range, which is highly unlikely unless the training data contains network activity from a custom protocol implemented on port 53213, which is unverifiable.

3.1.2.2 Time Dependent Network Flow Generation

Most network flow generation methods proposed in the state-of-the-art treat flows independently, without considering correlations among them. We consider this a major limitation in generating realistic network activities, as certain network activities result in multiple network flows. For example, before establishing an HTTP connection, a client may need to contact a DNS server to resolve the domain name of the requested website. Thus, a single action by the client can generate two flows, one to the DNS server and another to the website host. This area of improvement is noted by the survey study of Anande et al. [10], which summarizes the usage

[7] https://www.unb.ca/cic/datasets/nsl.html

of GAN for generating network traffic.

Xu et al. [19] are the first to address this issue with STAN (Synthetic network Traffic generation with Autoregressive Neural models), an autoregressive model predicting network flow features based on previous flows. An autoregressive model is a statistical model used in time series analysis that forecasts future values based on past observations. In the case of STAN, it uses past network flow data to predict future features, capturing temporal dependencies and patterns within the data. The model is trained using a CNN-based autoregressive neural network, which processes a sliding window of past data points with convolutional layers to extract relevant features, creating a context for predicting subsequent data points.

STAN predicts the distribution of network flow attributes for each data point. Continuous attributes are modeled using mixture density networks, which output a Gaussian mixture model, while discrete attributes are handled by softmax layers predicting the probability distribution over possible categories. Once trained, STAN generates new network flows by sequentially predicting each attribute, starting from the initial data points sampled from the learned marginal distribution and using the context of the previous k data points for subsequent predictions. Although the authors compare their method against Bayesian Networks with impressive results, the lack of maintenance on their GitHub repository[8] hinders the reproducibility of their work.

Yin et al. [47] critic the independent generation approach as a "strawman" method and propose NetShare, a model based on a sequential generation method called DoppelGANger [84]. NetShare generated sequences of network flows and packet headers. In this context, we are only interested in their proposal for generating network flow sequences. DoppelGANger is a GAN that generates a multivariate time series conditioned on metadata, consisting of two GAN: one generating the metadata and another, based on an RNN structure, generating the time series of network features conditioned on the generated metadata. For NetShare, the metadata included five-tuple - source and destination IP addresses, ports, and transport protocol - while the time series encompassed elements like packet size, number of packets, and timestamps.

Being trained on the UGR'16[9], TON_IoT[10], and CIDDS[11] datasets, NetShare's generated flows are assessed using statistical measures and their impact on anomaly detector training. It

[8] https://github.com/ShengzheXu/stan
[9] https://nesg.ugr.es/nesg-ugr16/
[10] https://research.unsw.edu.au/projects/toniot-datasets
[11] https://www.hs-coburg.de/forschung/forschungsprojekte-oeffentlich/informationstechnologie/cidds-coburg-intrusion-detection-data-sets.html

is currently the most up-to-date and well-maintained solution for generating network flows. However, some examples provided by the authors suggest that the overall quality of the generated traffic requires significant improvement (see Table 3.2). Therefore, generating realistic network flows with temporal dependencies remains an open challenge.

Src. IP	Dst. IP	Src. Port	Dst. Port	Protocol	Dur.	pkt	byt
42.220.247.236	42.219.159.106	31737	53018	TCP	0.02	1	80
37.192.66.96	42.219.153.89	46143	45255	UDP	0.01	1	155
58.167.198.248	42.219.153.155	35689	48250	TCP	0.10	2	137
194.199.153.251	42.219.146.48	45821	53	UDP	0.06	1	38
47.48.160.82	42.219.151.40	6000	20623	TCP	0.02	1	36
241.54.187.197	42.219.145.218	48005	80	TCP	0.10	1	39

Table 3.2: NetShare generated network flows. Those samples are extracted from the artifact given by the authors [47]. pkt: Packets, but: Bytes, Dur.:Duration. We can see that the first tree flows have ephemeral ports for both Source Port and Destination Port, which is highly unrealistic

3.1.2.3 Identified Gaps and Improvements

We, therefore, identify two potential avenues for contributing to state-of-the-art in the general-purpose generation of network flows:

- The first avenue involves challenging the dominance of GAN, which are used in four out of five studies on this topic. This can be achieved by exploring alternative methods, as has been successfully done in other fields, such as medical data generation [83]. The goal would be to determine whether more straightforward methods can outperform GAN in generating network flows.
- The second avenue focuses on developing methods for generating time-dependent network flows while maintaining realism at the level of individual flows. This would address a significant limitation in current methodologies, which often overlook the temporal dependencies between flows.

3.1.3 Generating Network Traffic in Other Data Format

While our thesis focuses solely on generating synthetic network flows, other formats of network traffic data have been generated using machine learning. We do not present these extensively here but give a brief overview that helps highlight the challenges associated with using

AI to generate data types other than tabular data. The two other relevant formats of traffic generated in the state of the art are packet payloads and sequences of headers. Although our study does not directly compare to these methods, we include them for general knowledge.

3.1.3.1 Generation of Packet Payloads

Network packet generation aims to produce the content of individual packets. We are not aware of any method tackling this issue differently than using neural networks. The main challenge lies in accurately representing the bytes of a packet for generation by neural networks. A critical aspect is transforming the binary content of a payload into a continuous input suitable for neural networks. Two methods of encoding the payload are currently being used. We present them here briefly, and we illustrate them in Figure 3.4:

- **Encoding Packet Data as Images**: Chen et al. [85] represent the packet payload as a grayscale image, leveraging the pattern recognition capabilities of Convolutional Neural Networks (CNNs) to generate realistic packets with their model, PAC-GAN. Initially, packet bytes were represented as grayscale levels in an image. However, this method failed to capture the necessary details for generation, prompting the authors to repeat each byte multiple times within the image, creating regional patterns that exploited the CNN's ability to capture local spatial dependencies. Compared to image generation, the challenge with packet generation lies in the precision required: slight variations in a byte can significantly alter its meaning, unlike small changes in a pixel of a grayscale image. To mitigate this, PAC-GAN generates ranges of values instead of precise byte values, splitting each byte into half-bytes (nibbles) and assigning them to specific sub-ranges. This ensured that the generated pixel values corresponded to the correct nibble subrange. Packets are encoded in 28x28 grayscale images, padded, or truncated as necessary. Meddahi et al. [86] apply a similar encoding scheme to generate SIP traffic with SIP-GAN, a modified DCGAN, incorporating the average byte error rate in the SIP Request Line as a part of the training loss of the generator.

- **Encoding Packet Data as Sequences of Discrete Values and Transforming Them Using One-Hot Encoding**: Nukavarapu et al. [87] propose MiragePckt, a DNS packet generator using a sequence-based model. The generator comprises a 1D CNN and softmax function, reproducing sequences of one-hot encoded vectors, which are then mapped back to their byte values. This approach simplifies the encoding process and enhances the scalability for large byte streams.

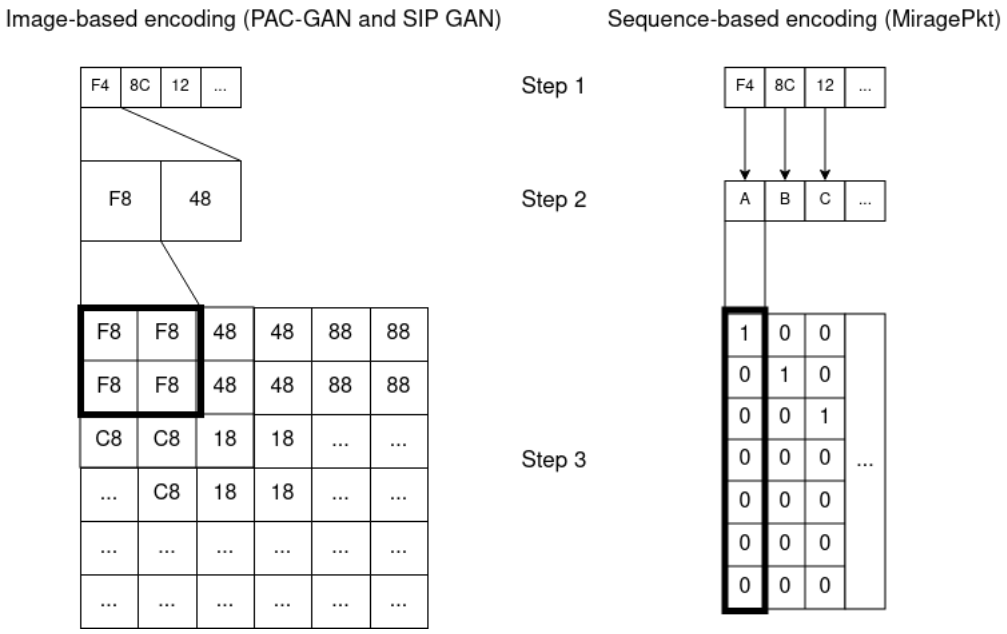

Figure 3.4: Difference of encoding-scheme between PAC-GAN/SIP-GAN and MiragePkt. In both cases, the goal is to transform a sequence of bytes into a matrix that can be given as input to a neural network. The option on the left is to transform the sequence in an image with a local region encoding nibble. Each byte is separated into two nibbles, and each nibble becomes a byte by adding 0x8 at the end, and this byte is repeated in a square region of the resulting image. The option on the right is to consider the bytes as discrete values in an alphabet and then encode their one-hot representation as a matrix column.

Overall, the difficulty of packet generation using AI methods revolves around the encoding problem: how to transform the binary content of a payload into a continuous input suitable for neural network-based methods. The complexity of this challenge has led authors to generate only a tiny fraction of the potential traffic.

3.1.3.2 Generation of sequences of headers

Unlike PAC-GAN, SIP-GAN, and MiragePkt, which focus on generating packet payloads and individual packet content, the models discussed in this Subsection address the generation of sequences of packet headers.

Dowoo Baik et al. [88] aims to generate the sequence and timing of packet headers in the communication stream with their model, PcapGAN. This model recreates header sequences by training three GAN on traffic extracted from pcap files. The communication is represented by directed graphs, where nodes correspond to IP addresses and edges to packets exchanged between hosts. Inter-arrival times are encoded as grayscale images, and header fields are encoded as sequences of discrete tokens. These three elements represent traffic and are generated by three different GAN.

The first step of generation involves using an encoder to create a style vector representing the traffic graph. This style vector conditions the generation of the three GAN. GraphGAN generates new communication graphs, SeqGAN generates new header sequences, and StyleGAN generates new inter-arrival grayscale images. The generated elements are then recombined to produce synthetic pcap files. The different processes of generation of PcapGAN are illustrated in Figure 3.5. Quality is assessed using Wireshark and Principal Component Analysis to compare generated and real network flows, with no significant discrepancies found between the two.

Meslet-Millet et al. [89] propose NeCSTGen, which uses Variational Autoencoders (VAE) to encode header features into a latent space. Once the VAE well represents the header features in this latent space, they identify the cluster header characteristics using a Gaussian Mixture Model (GMM). From there, the authors can generate new sets of header features, but the problem of sequencing such headers remains open. To find a way to order their different headers in traffic, they train a Recurrent Neural Network (RNN) to generate the sequence of ordering based on the headers sampled in the previous step. The authors include a GitHub link to their tool[12]. Still, they only compare it to traditional and older network traffic generators such as HARPOON [90] and LitGen [91], and not against other ML methods.

[12] https://github.com/fmeslet/NeCSTGen

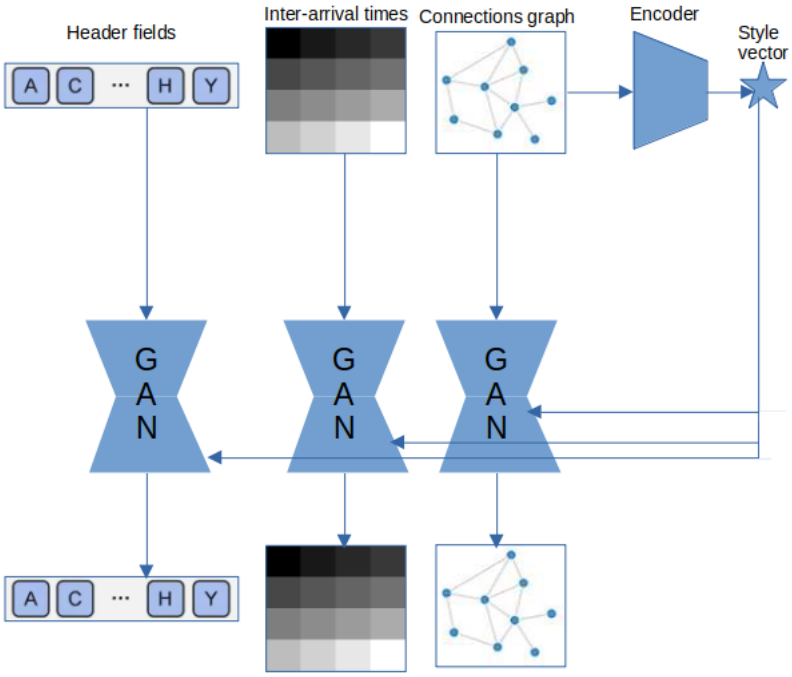

Figure 3.5: Data generation pipeline of PcapGAN. Three objects represent the traffic: 1) A sequence of discrete tokens for the header fields, 2) A grey-scale image for the inter-arrival times, and 3) A directed graph for the connections. The first step is to encode the graph in a style vector. This style vector conditions the generation of the GAN of the different elements.

In contrast, Jiang et al. [18] encode entire sequences of headers as images using nPrint, converting traffic into standardized bits. Each bit is color-coded (green for set, red for unset, gray for vacant) to represent traffic visually. Grouping packets into sets of 1024, each row represented a packet header. The NetDiffusion model, a U-net[13] based diffusion model, generates new traffic images. Comparisons with other synthetic traffic generation methods like NetShare [47] and DoppelGANger [84] show statistical similarities and comparable performance in anomaly-detection models trained on both authentic and generated data.

Moving beyond the network flow format presents unique challenges, especially in how data is represented and encoded. We face two main issues at the packet level: generating the packet content and determining their correct ordering. The complexity of solving these two tasks, let alone combining them in a single-generation process, remains largely unsolved. In addition to the reasons mentioned in Section 1.3 for focusing on network flow generation, we decide to concentrate on network flows due to the larger body of work in this format and the relatively early stage of synthetic network generation at the packet level.

3.1.4 Limitations of current synthetic network traffic generation

Despite significant advancements in generating synthetic network traffic flows using AI-based methods, several limitations hinder these techniques' broader applicability and effectiveness.

3.1.4.1 Model Specialization and Lack of Generalization

The generative methods discussed, particularly those that enhance the performance of a specific NIDS, are highly specialized. These methods are typically tailored to specific NIDS, resulting in poor generalization beyond their intended application. The close coupling between the generative models and the particular characteristics of the NIDS complicates the application of these techniques to the broader goal of general network traffic generation. This greatly reduces their utility in our objective of constituting a synthetic evaluation dataset for general NIDS evaluation.

3.1.4.2 Handling Temporal Dependencies

Most existing models treat flows independently without considering their temporal dependencies. This oversight is a significant limitation because network activities often generate mul-

[13] A U-Net is a convolutional neural network with a U-shaped architecture, combining downsampling and upsampling paths with skip connections, allowing for detailed pixel-level predictions

tiple correlated flows, such as DNS queries followed by HTTP requests. The inability to model these dependencies accurately reduces the realism of the synthetic traffic, which could impact its effectiveness in our objective of NIDS evaluation.

3.1.4.3 Mode Collapse and Feature Dependency Learning in GAN

Although GAN have shown promising capabilities in generating synthetic network flows, they suffer from joint issues like mode collapse, where the generator produces limited diversity in the synthetic data. Furthermore, GAN struggle with learning dependencies between different features in tabular data, leading to unrealistic correlations in the generated traffic. For example, anomalies like ephemeral ports appearing as both source and destination ports, which are highly unlikely in real datasets, highlight the limitations in the current generative models' ability to capture feature dependencies accurately.

3.1.4.4 Challenges in Packet Payload Generation

Generating packet payloads using AI methods faces significant challenges due to the complexity of encoding binary content into formats suitable for neural networks. Existing approaches, such as encoding packet data as images or sequences of discrete values, often result in simplifications that do not allow traffic generation from a wide range of applications. On top of that, generating a payload based on the ordering of its packet in the communication remains unsolved.

3.1.4.5 Reproducibility and Lack of Comprehensive Comparisons

Some proposed methods, like STAN, face issues related to reproducibility due to a lack of maintenance and updates in their implementation repositories. Moreover, there is a noticeable gap in the literature concerning comprehensive comparisons between different generative approaches. While GAN are widely used, other methods like Bayesian Networks have shown potential in similar contexts (e.g., medical data generation [83]). Still, little has been explored of their effectiveness in network traffic generation. This lack of comparative analysis limits the ability to identify the most effective techniques for specific use cases.

3.2 Quality evaluation of generated traffic

To develop a methodology for comparing network flow generation methods and validating the quality of any proposed model, it is essential to establish a systematic approach for evaluating the generated data. The aim is to transition from subjective, qualitative assessments to objective, quantitative measurements that answer the question, "How good is the quality of the generated traffic?" Achieving this requires addressing two core questions: What defines high-quality network flow generation, and how can it be effectively measured?

As stated in Section 2.1, the generation of network flow data can be considered close to tabular data generation. Consequently, we can draw inspiration from how tabular data generation is evaluated in other contexts to identify common properties and metrics. This section first presents the methods used to assess tabular data generation, and afterward presents methods used for evaluating network flow generation.

3.2.1 Evaluating Tabular Data Generation

Based on the problem formulation by Livieris et al. [92], tabular data generation involves working with tables consisting of multiple variables, denoted as $(X_1, X_2, \ldots, X_n)$. The task of generating synthetic data entails creating new samples, which are new combinations of these variables, essentially forming new rows in the table.

After a tabular data generation process, we have two tables of data: $(X_1, X_2, \ldots, X_n)_{\text{real}}$, representing the real data table, and $(X_1, X_2, \ldots, X_n)_{\text{generated}}$, representing the synthetic data table. A synthetic sample is defined as a row in the synthetic table, i.e., a specific combination of the variables $(X_1, X_2, \ldots, X_n)_{\text{generated}}$.

Evaluating the quality of synthetic data generation involves comparing the distributions of the actual variables with those of the generated variables. This comparison helps determine how well the synthetic data replicates the properties and patterns of the real data.

3.2.1.1 Primary criteria of quality evaluation

Hernandez et al. [93] propose three primary criteria for evaluating the quality of synthetic data generation: resemblance, utility, and generalization. **Ressemblance** assesses how well the distribution of the synthetic data aligns with the actual data distribution. **Utility** measures the usefulness of the synthetic data for various analytical tasks, including training machine learning models. **Generalization** evaluates the model's capability to produce new, unseen samples

rather than merely replicating the training data.

Meanwhile, Dankar et al. [94] categorize resemblance and utility under the broader term "Utility" further divided into two sub-criteria: *application fidelity* (similar to Hernandez et al.'s concept of utility) and *broad fidelity* (similar to resemblance). Dankar et al. do not address generalization; however, this is done by Gonçalves et al. [83] while keeping the broad "Utility" criterion of Dankar et al.

An interesting point in the work of Dankar et al. is their approach to evaluate broad fidelity (resemblance) on three levels: attribute, bivariate, and population. This involves assessing the resemblance of the marginal distributions, the conditional distributions, and the joint distributions between the real and synthetic data $(X_1, \ldots, X_n)_{\text{generated}}$ and $(X_1, \ldots, X_n)_{\text{real}}$.

Integrating insights from these papers, we see three core criteria for evaluating the quality of tabular data generation: resemblance, utility, and generalization. Resemblance, however, is nuanced and evaluated at three levels: marginal distribution, conditional distribution, and joint distribution.

Two key issues arise in evaluating synthetic data quality. First, utility often overlaps with resemblance; when synthetic data closely mirrors the real data distribution, machine learning models trained on it are likely to perform similarly to those trained on real data. Second, as Naeem et al. [95] observe, resemblance metrics frequently struggle to distinguish between whether synthetic samples appear realistic and whether the generation captures the full variability of real data. They suggest refining resemblance into two distinct criteria: **realism** (termed fidelity in their paper but renamed here for consistency with Dankar et al.'s terminology) and **diversity**. Realism assesses whether synthetic samples are drawn from the same distribution as the real data, while diversity examines whether the synthetic dataset maintains a similar level of variability as the real data.

These criteria, realism and diversity, are not distinguishable when evaluating similarity at the marginal distribution level. However, distinguishing them at the joint distribution level helps diagnose generative model issues such as mode collapse (low diversity, where only a fraction of the actual distribution is captured in the synthetic data) and mode invention (low Realism, where generated data do not align with the real data distribution). This realism/diversity duality has been adopted in several works [95]–[97].

Table 3.3 summarizes the different criteria names used across various papers. Ultimately, we identify four essential criteria: realism, diversity, utility, and novelty. This naming of the four criteria—realism, diversity, utility, novelty—is purely from our own, but the same criteria are named differently in the literature. Once those criteria are named and defined, it's essential

	Realism	**Diversity**	**Utility**	**Novelty**
[93]	Ressemblance		Utility	Privacy
[94]	Utility (broad fidelity) (application fidelity)			*None*
[83]	Utility			Information Disclosure
[98]	*None*	*None*	*None*	Generalization
[95], [96]	Fidelity	Diversity	*None*	*None*
[99]	Fidelity	diversity	*None*	Generalization
[97]	Fidelity	fairness	*None*	Generalization/Bias

Table 3.3: Different naming of the various criteria. There are basically four criteria that we named realism, diversity, utility, and novelty. This table refers to all the different names that have been given to those four criteria across the bibliography.

to delve deeper into the evaluation of each one of them. One key aspect of both realism and diversity is that they can be evaluated at marginal, conditional, and joint levels. We, therefore, introduce tools to evaluate these criteria in those three levels before enumerating tools to asses the two other criteria: utility and novelty.

3.2.1.2 Evaluation of Realism and Diversity

The goal of this evaluation is to assess the similarity between the distributions $(X_1, \ldots, X_n)_{\text{real}}$ and $(X_1, \ldots, X_n)_{\text{generated}}$. We group the evaluation of realism and diversity together because many metrics used in tabular data evaluation measure both (see Table 3.4). However, where possible, we indicate when a metric allows us to differentiate between these two criteria.

Evaluation of Marginal Distributions The first step in evaluating the resemblance between the real and generated data distributions is to assess the marginal distributions of each variable X_i for the real and generated data across all i. This approach indicates how well the generated data matches the real data but does not fully differentiate between realism (how closely the generated data resembles real data) and diversity (whether the generated data covers the total variability of the real data).

Given that tabular data typically includes both categorical and numerical features, different methods are often used to evaluate these types of data:

- **Categorical Data**: The Kullback–Leibler divergence (KLD) is commonly used to measure the similarity between the real and generated distributions for categorical data [83], [94]–[97]. The KLD, denoted as $D_{KL}(P\|Q)$, quantifies the divergence bethat enhance the per-

formance of a specific NIDStween two probability distributions P (real) and Q (generated). It is calculated as:

$$D_{KL}(P\|Q) = \sum_{x} P(X_i = x) \log \left(\frac{P(X_i = x)}{Q(X_i = x)} \right)$$

where x represents the different categories of the categorical variable X_i, and $P(X_i = x)$ and $Q(X_i = x)$ are the probabilities of category x under the distributions P and Q, respectively.

Some studies [19], [47] prefer using the symmetrized and smoothed version, Jensen-Shannon Divergence (JSD), or the square root of the JSD, known as the Jensen-Shannon distance [93]. The JSD is defined for two probability distributions P and Q and a midpoint distribution $M = \frac{1}{2}(P + Q)$. The JSD is calculated as:

$$JSD(P\|Q) = \frac{1}{2} D_{KL}(P\|M) + \frac{1}{2} D_{KL}(Q\|M)$$

where M is the average of the distributions P and Q. The JSD is particularly useful for measuring similarity because it is always finite and symmetric.

- **Numerical Data**: The Wasserstein Distance (WD), also known as the Earth Mover's Distance, is the most frequently used metric for numerical data [92], [93], [96]. It measures the distance between two one-dimensional numerical distributions $P(X_i)$ and $Q(X_i)$. The WD is defined as:

$$W(P, Q) = \int_{-\infty}^{\infty} |F_P(x) - F_Q(x)| \, dx$$

where F_P and F_Q are the cumulative distribution functions (CDFs) of $P(X_i)$ and $Q(X_i)$, respectively. The WD provides a meaningful measure of the difference between distributions, especially for numerical data.

An alternative for evaluating numerical features is the Kolmogorov-Smirnov (KS) test, utilized by some papers [94], [100]. The KS test is a non-parametric test that compares the CDFs of two distributions, F_P for the real data and F_Q for the generated data. For a numerical feature X_i, the KS statistic is defined as:

$$D = \sup_{x} |F_P(x) - F_Q(x)|$$

where $\sup_x$ denotes the upper bound over all values x of the feature X_i. This test measures the maximum absolute difference between the CDFs, providing an evaluation of how well the generated data matches the real data for the feature X_i.

However, evaluating only the marginal distributions is insufficient for establishing the overall resemblance between the generated and real data. In addition to the marginal distribution, evaluating how well the generated data captures the relationships between variables is essential, which requires examining the conditional distributions.

Evaluation of Conditional Distributions The objective of evaluating the conditional distribution is to ensure that the synthetic distribution $(X_1, X_2, \ldots, X_n)_{\text{generated}}$ captures the statistical dependency structure of the original data. As with the evaluation of marginal distributions, different techniques are used for categorical and numerical features [93].

For numerical features, the primary metric used in the state of the art is the Pairwise Correlation Difference (PCD) [83], [94], also referred to as Pairwise Pearson Correlation (PPC) in [93]. PCD measures the L_2 norm of the difference between the real data's correlation matrix and the generated data's correlation matrix. The Pearson correlation coefficient is utilized to construct these correlation matrices. The PCD is calculated as follows:

$$PCD = \|\text{Corr}_{\text{real}} - \text{Corr}_{\text{generated}}\|_2$$

Where $\text{Corr}_{\text{real}}$ and $\text{Corr}_{\text{generated}}$ are the correlation matrices of the real and generated data, respectively.

For categorical features, Hernandez et al. [93] suggest evaluating the dependency structure by examining the contingency matrices for each pair of categorical features in both the real and generated data. The metric used is the Contingency Matrix Difference (CMD), which is calculated by summing the differences between the real and generated contingency matrices. CMD is defined as:

$$CMD = \sum_{i,j} |C_{real}(i,j) - C_{generated}(i,j)|$$

Where $C_{real}(i,j)$ and $C_{generated}(i,j)$ represent the contingency matrices of the real and generated data for categories i and j.

These metrics help assess how well the synthetic data preserves the relationships and dependencies present in the original data, which is crucial for maintaining the integrity and usability of the synthetic data.

Evaluation of Joint Distributions Beyond evaluating marginal and conditional distributions, it is essential to assess the joint distribution to capture the relationship between all variables fully. The goal here is to evaluate whether the joint distribution of $(X_1, X_2, \ldots, X_n)_{\text{generated}}$ is similar to that of $(X_1, X_2, \ldots, X_n)_{\text{real}}$, i.e.,

$$\begin{aligned} &\forall(\omega_1, \omega_2, \ldots, \omega_n) \in \Omega, \\ &P\left((X_1, X_2, \ldots, X_n)_{\text{generated}} = (\omega_1, \omega_2, \ldots, \omega_n)\right) \\ &= P\left((X_1, X_2, \ldots, X_n)_{\text{real}} = (\omega_1, \omega_2, \ldots, \omega_n)\right). \end{aligned} \tag{3.1}$$

A method to assess this similarity is *LogCluster* [83], [94]. LogCluster measures the similarity of the underlying latent structure of the real and synthetic datasets in terms of clustering. The real and synthetic datasets are first merged into a single dataset to compute this metric. Then, a cluster analysis is performed on the merged dataset using the k-means algorithm with a fixed number of clusters G. The metric is calculated as:

$$U_c((X_1, \ldots, X_n)_{\text{real}}, (X_1, \ldots, X_n)_{\text{generated}}) = \log\left(\frac{1}{G}\sum_{j=1}^{G}\left[\frac{n_j^R}{n_j} - c\right]^2\right),$$

Where n_j is the number of samples in the j-th cluster, n_j^R is the number of samples from the real dataset in the j-th cluster, and $c = \frac{n^R}{n^R+n^S}$ with n^R and n^S being the total number of real and synthetic samples, respectively. Large values of U_c indicate disparities in cluster memberships, suggesting differences in the distribution of real and synthetic data.

An alternative to LogCluster is the *Propensity Score*, as utilized by Dankar et al. [94]. The real and synthetic datasets are combined to calculate the propensity score, and a binary indicator is assigned to each record (1 for synthetic rows and 0 for original rows). A binary classification model is trained to discriminate between real and synthetic records. The model's predicted values (propensity scores $\hat{p}_i$) are then used to compute the metric:

$$pMSE = \frac{1}{N}\sum_i (\hat{p}_i - 0.5)^2,$$

Where N is the size of the combined dataset. Propensity scores range from 0 to 0.25, with 0 indicating no distinguishability between the two datasets, suggesting perfect overfitting by the generator, while 0.25 indicates complete distinguishability. Like marginal evaluation, propensity score and LogCluster do not allow us to differentiate between realism and diversity.

On the other hand, *Precision and Recall*-based methods uniquely differentiate between the criteria of realism and diversity. They measure the similarity of generated instances to real ones

(precision) and the ability of a generator to synthesize all cases found in the real set (recall) [97].

Similarly to Precision and Recall, Naeem et al. [95] propose *Density and Coverage* to address outliers in evaluation. Realism is assessed by Density, which counts how many real-sample neighborhood spheres contain each synthetic sample, calculated using k-nearest neighbors. A low Density score indicates a lack of proximity between real and synthetic data. Coverage assesses diversity by counting the number of synthetic neighborhood spheres that include each real sample. A low Coverage score suggests that several real samples lack synthetic counterparts in their vicinity, indicating insufficient variance capture in the synthetic distribution.

Finally, Alaa et al. [99] introduce α-*Precision* and β-*Recall*, which characterize the realism and diversity power of generative models. These metrics assume that a fraction $1 - \alpha$ and $1 - \beta$ of the real and synthetic data, respectively, are "outliers", while α and β are "typical". α-Precision is the fraction of synthetic samples that resemble the "most typical" α real samples, whereas β-Recall is the fraction of real samples covered by the most typical β synthetic samples. These metrics are evaluated across all $\alpha, \beta \in [0, 1]$. Data points are embedded into hyperspheres, with most samples concentrated around the centers, typical samples near the centers, and outliers near the boundaries.

3.2.1.3 Evaluation of Utility

The utility criterion for evaluating synthetic data focuses on the performance of models trained on synthetic data in real-world tasks. It assesses whether synthetic data can effectively capture essential features and patterns, making it a viable substitute for real data in various applications.

To evaluate utility, both synthetic and real data are used in specific machine learning tasks, and any discrepancies in performance are analyzed. The most widely used methodology for this evaluation is *Train on Synthetic, Test on Real* (TSTR) [92], [93], [101]. In TSTR, a machine learning model is trained on a specific task using the synthetic dataset and evaluated on a held-out real test set. This model's performance metrics (such as accuracy, precision, and F1 score) are compared to those of a similar model (with the same structure) trained on real data. The TSTR metric provides insights into how well the synthetic data captures the underlying patterns and distributions of the real data. Suppose a model trained on synthetic data performs comparably to one trained on real data when tested on a real dataset. In that case, it indicates a high resemblance between the synthetic and real data.

Another approach involves evaluating the impact of injecting synthetic data into the training set of a machine learning classifier [71], [73]. In this method, one classifier is trained solely

on real data, while the other classifier of the same model architecture is trained on a mixture of real and synthetic data. The performances of these classifiers are then compared. A negligible or irrelevant drop in performance due to the inclusion of synthetic data suggests that the synthetic data closely resembles the real data.

A challenge highlighted by Hernandez et al. [93] is that the effectiveness of utility measures for synthetic data can depend on the type of machine learning model used. Different models vary in their ability to capture dependencies and nuances within the data. A more sophisticated model might detect subtle discrepancies between synthetic and real data that a simpler model might overlook. Therefore, the choice of model can significantly influence the evaluation outcome, as it determines the model's sensitivity to differences between training on synthetic versus real data.

3.2.1.4 Evaluation of Novelty

Evaluating the novelty of a generative model involves assessing whether the model produces unique samples rather than simply replicating the training data. The goal is to determine if the generated samples reflect the data distribution's full range rather than mere reproductions or slight modifications of training examples.

Meehan et al. [98] evaluate the novelty of a generative model by examining whether the generated samples are systematically closer to the training data than independently drawn samples from the same distribution. They apply a non-parametric test that measures the distance between generated samples and their nearest training samples, comparing it to the distances between test samples and the training set. If the generated samples are found to be closer to the training samples than the test samples, it indicates that the model may be overfitting, memorizing the training data instead of capturing the broader diversity of the data distribution. To gain deeper insights into the model's behavior, they divide the data space into smaller regions or cells and conduct the novelty tests within each region. This allows them to detect if overfitting or data-copying occurs in specific areas rather than across the entire dataset. By combining the results from all regions, they obtain a comprehensive assessment of how much the model is memorizing training data, providing a detailed evaluation of the model's ability to generate truly novel samples.

Similarly to this approach, which assess the proximity of generated samples to both training and test data, Gonçalves et al. [83] propose the *Membership Disclosure* (MD) method to evaluate novelty by identifying whether synthetic samples closely replicate real data samples. The MD method operates by analyzing the Hamming distances between each real sample (from

both the training and testing sets) and all synthetic samples.

For each real sample, its Hamming distance to synthetic samples is used to classify it as either training or testing data. If a synthetic sample is found to have a very low Hamming distance to a real sample, there is a plausible indication that this real sample, if from the training set, may have been "leaked" into the synthetic data, suggesting overfitting. Gonçalves et al. use various thresholds of Hamming distance as classifiers: if the distance between a real sample and a synthetic sample falls below a given threshold, the real sample is classified as belonging to the training set. The outcomes of this classification (true positives, false positives, true negatives, and false negatives) allow for calculating precision and recall. High precision and recall indicate that the method effectively distinguishes between synthetic and real training samples. If multiple classifiers consistently detect training data with high accuracy, it suggests that the synthetic samples are closely aligned with the training data, highlighting potential overfitting.

While the non-parametric test provides an overall view of distance distributions, the Membership Disclosure method offers an instance-level assessment, giving a more granular evaluation of potential overfitting and direct replication in synthetic data generation.

In addition to these methods, Alaa et al. [99] propose an *authenticity score* to quantify how much a generative model creates novel samples. The score is derived from a mixture model representing the generative process as a combination of truly new synthetic samples and slightly varied versions of the training data. An authenticity classifier is employed to assess whether a sample is memorized or novel, using a likelihood-ratio test that compares the distance of synthetic samples to their nearest training counterparts. The classifier evaluates if a synthetic sample is closer to the nearest training data point than to other points, thus identifying overfitting and lack of generalization. This approach provides a comprehensive framework for distinguishing between genuine generalization and simple data memorization.

Gonçalves et al. [83] used an *Attribute Disclosure* metric to assess how easily a specific feature in the real data can be inferred by examining similar synthetic data. This metric is primarily designed to preserve privacy in the training data. It is particularly useful when the goal is to prevent individuals with access to certain network features from reconstructing private or sensitive information. However, this privacy-focused metric is unrelated to the task of generating synthetic data that closely mimics real data for the purpose of evaluating novelty.

In Table 3.4, we reference all the metrics used to evaluate tabular data generation as well as the criteria they assess, the data type they are working with, and the distribution on which they apply.

Score	Criteria				Description of the score				
	Real.	Div.	Util.	Nov.	Distribution	Input		Description	Ref.(s)
						Num.	*Cat.*		
KLD	✓	✓			Marg. Distr.		✓	Compute the KLD between generated and real feature	[83], [94]–[97]
JSD	✓	✓			Marg. Distr.		✓	Compute the JSD between generated and real feature	[19], [47]
WD	✓	✓			Marg. Distr.	✓		Compute the WD between generated and real features	[92], [93], [96]
KS test	✓	✓			Marg. Distr.	✓		Non-parametric test that compares the CDFs of generated and real distributions	[94], [100]
PCD	✓				Cond. Distr.	✓		Compare the pairwise correlation matrix of the generated data with the real one	[83], [93], [94]
CMD	✓				Cond. Distr.		✓	Differences between the real and generated contingency matrices	[93]
LogCluster	✓	✓			Joint Distr.	✓	✓	Proximity of real and generated samples, evaluated through clusterization	[83], [94]
Propensity	✓				Joint Distr.	✓	✓	Probability that a data record is synthetic rather than real	[94]
Precision	✓				Joint Distr.	✓	✓	Probability that a generated sample belongs to the real distribution	[97]
Recall		✓			Joint Distr.	✓	✓	Probability that a real sample belongs to the generated distribution	[97]
Density	✓				Joint Distr.	✓	✓	Probability mass of the generated distribution covered by the real distribution	[95]
Coverage		✓			Joint Distr.	✓	✓	Probability mass of the real distribution covered by the generated distribution	[95]
α-precision	✓				Joint Distr.	✓	✓	Density score applied to an hypersphere embedding of the data manifold	[99]
β-recall		✓			Joint Distr.	✓	✓	Coverage score applied to an hypersphere embedding of the data manifold	[99]
TSTR			✓		Joint Distr.	✓	✓	Comparison of a model trained on real data to one trained on synthetic data	[92], [93], [101]
ML training			✓		Joint Distr.	✓	✓	Impact of injecting synthetic data into the training set of a ML classifier	[71], [73]
Test for copy				✓	Joint Distr.	✓	✓	Non-parametric test that compares distances between generated and training data	[98]
Authenticity				✓	Joint Distr.	✓	✓	Modification of Test for copy but with a likelihood-ratio test	[99]
MD				✓	Joint Distr.		✓	Detection of training sample copied inside the generated set	[83]

Table 3.4: Summary of the main evaluation methods used to assess the quality of synthetic tabular data and the criteria they assess.

3.2.2 Evaluating Network Traffic Generation

The previous Subsection expanded on the evaluation of tabular data generation. This Subsection specifically focuses on the evaluation of network traffic generation. We discuss how tabular metrics are adapted for this context, the current network traffic generation evaluation issues, and the specific criteria for assessing it. Table 3.5 summarizes the different metrics used by Network Flow generation research.

3.2.2.1 Application of Tabular Data Evaluation

Many papers apply metrics commonly used for tabular data generation to evaluate network traffic generation [19], [46], [47], [80], [84]. This is particularly relevant when the generated network data is in a network flow format [19], [46], [47], [80]. For instance, Jensen-Shannon Divergence (JSD) has been used to evaluate the realism, and diversity of generated traffic [19], [47]. The Wasserstein Distance (WD) is another frequently used metric [46], [47], [80], [84]. Additionally, a χ^2 test was utilized by Bourou et al. [46] to assess the goodness-of-fit for categorical distributions. Correlation studies have also compared the correlation structures in the generated and real data. For example, [46], [84] used PCD to ensure that the generated network flows preserve the correlation structure of the real network flows.

3.2.2.2 Abuse of the Utility Metric

Many studies evaluate synthetic data by examining its impact on the performance of NIDS [19], [46], [47], [73], [84], underscoring the popularity of utility evaluation for comparing synthetic network flow generation methods. Zingo et al. [101] introduced the Train on Synthetic, Test on Real (TSTR) approach to measure how well-generated traffic serves specific applications, like NIDS enhancement. While TSTR and similar methods provide insights into utility for specific tasks, they may not fully capture synthetic data's general quality or its applicability across diverse models and use cases.

For example, Xu et al. [73] evaluate traffic generated by WGAN-GP based solely on performance improvements in a custom classifier, which, while useful, may not generalize to other models or scenarios. As Hernandez et al. [93] observe, utility evaluations tied to particular models can obscure synthetic data's overall quality. While utility metrics are essential for understanding data effectiveness in particular contexts, they alone may not indicate broader data quality or its usability across varied applications.

3.2.2.3 Lack of Novelty Awareness

Most papers on synthetic traffic generation do not address the potential for models to memorize and replicate training data. This concern is highlighted by Yien et al. [47], who noted that neither their work nor other previous studies [84] provided methods to evaluate the potential privacy leakage of generative models. Instead, they explored the fidelity-privacy tradeoff by assessing the impact of Differential Privacy (DP) on model training. While this approach focuses on privacy guarantees, it does not directly address the issue of novelty, which involves assessing whether generated samples are mere copies of the training data. The evaluation of novelty is crucial for avoiding overfitting and ensuring that the model can generalize beyond the training set. This aspect remains largely unexplored in the state-of-the-art synthetic traffic generation approaches.

3.2.2.4 Compliance: A Network-Specific Criterion

Besides the above factors of data quality evaluation, another critical aspect of evaluating generated network traffic is ensuring that the data adheres to network protocol specifications. Ring et al. [16], who generate traffic through network flows, assess each new flow by verifying its adherence to network rules. For example, they ensured that flows using the UDP protocol did not include TCP flags, referring to this evaluation as passing network flows through a series of Domain Knowledge Tests. This approach is also utilized in evaluating STAN, the model presented by Xu et al. [19].

In the context of packet traffic generation, models like PAC-GAN [85] and SIP-GAN [86] are evaluated by broadcasting the generated packets in a network and observing if the targets responded correctly according to protocol specifications. Additionally, the number of incorrectly generated field values in packets (i.e., non-compliance with network standards) are counted.

These evaluations reflect the importance of compliance as a criterion for good network traffic generation. Compliance measures how well the generated samples align with network protocol specifications. As Ring et al. [16] noted, while compliance and realism are related, they are distinct criteria. Realism evaluates how closely synthetic samples resemble actual data distributions, whereas compliance checks whether samples conform to network protocol specifications. A synthetic sample can appear highly realistic, resembling real data, yet fail to meet compliance standards. For instance, a packet might have a payload nearly identical to a real one but contains slight variations in critical bytes, which could invalidate the entire packet.

Score	Criteria					Description	
	Real.	Div.	Util.	Nov.	Comp.	Description	Ref.(s)
JSD	✓	✓				Compute the JSD between generated and real feature	[19], [46], [47], [80], [84]
WD	✓	✓				Compute the WD between generated and real feature	[46], [47], [80], [84]
KS Test	✓	✓				Non-parametric test that compares the CDFs of generated and real distributions	[46]
χ^2 Test	✓	✓				Probability that real and generated features are sampled from the same distributions	[46]
PCD	✓					Compare the pairwise correlation matrix of the generated data with the real one	[46], [84]
ML training			✓			Impact of injecting synthetic data into the training set of a ML classifier	[19], [46], [47], [73], [84]
TSTR			✓			Comparison of a model trained on real data to one trained on synthetic data	[101]
DKC					✓	Test the conformity of synthetic data to network specifications	[16], [19]
Network response					✓	Test the response of the network when synthetic data is broadcasted	[85], [86]
Byte Error Rate					✓	Number of bytes incorrectly generated	[85], [86]

Table 3.5: Summary of the function used to evaluate generated Network Traffic.

3.2.2.5 Evaluating the Preservation of Temporal Dependencies

As discussed in Subsection 3.1.2, several studies [19], [47], [84] emphasize the importance of generating network flows that preserve the dependencies between them over time. To evaluate whether their DoppelGANger model preserves these temporal dependencies, Lin et al. [84] compare the autocorrelation of the real-time series with that of the generated time series, averaging the differences across all time lags. Autocorrelation measures how the current value of a time series is related to its past values at various time intervals.

However, there are two main issues with Lin et al.'s evaluation approach. First, it only considers the preservation of temporal dependencies in the numerical features of the time series, such as the number of *Bytes* or *Packets* exchanged. It ignores the temporal dependencies in categorical features, such as *Port* or *Protocol*. In the context of network flows, this means that essential sequences, like an HTTPS connection being preceded by a DNS request, might be overlooked by this evaluation method. Second, by averaging the differences in autocorrelation across all possible time lags, the method risks smoothing out significant differences that occur at specific, relevant time lags. In time series analysis, not all time lags are equally important, and averaging can mask the preservation (or lack thereof) of dependencies at critical intervals.

STAN, on the other hand, is only evaluated by comparing the correlation of numerical features at time t with those at time $t-1$, which corresponds to an autocorrelation with a lag of 1. This approach is insufficient for detecting more distant dependencies, which is particularly problematic in network traffic analysis, where different interactions might produce flows that are not directly sequential. NetShare, meanwhile, does not assess the preservation of temporal dependencies at all, although the preservation of temporal dependencies is a vital problem of the paper.

In summary, the current methods for evaluating the consistency of temporal dependencies in numerical features are underdeveloped, and no existing work has proposed a solution to assess temporal dependencies in categorical features within the network domain.

3.2.3 Limitations of current synthetic traffic evaluation

Despite the advancements in evaluating network traffic generation, several limitations persist in the current methodologies.

3.2.3.1 Utility Metric Limitations

The widespread use of utility metrics, particularly in the context of training NIDS, highlights a significant limitation. Utility evaluations, such as Train on Synthetic and Test on Real (TSTR), depend highly on the specific machine learning models used in the assessment. This dependence makes it difficult to generalize the quality of synthetic data beyond the particular use case. As a result, while these evaluations can provide insights into the data's applicability for specific tasks, they may not comprehensively evaluate its overall quality or suitability for broader applications.

3.2.3.2 Lack of Comprehensive benchmarks for All Aspects of Tabular Data Generation

Independent network flow generation is a specialized form of tabular data generation, yet many studies fail to incorporate the recent advancements in evaluating tabular data generation comprehensively. For instance, novelty—the ability to generate original data rather than replicating the training set—is seldom assessed. A valuable contribution would be the development of a comprehensive benchmark that evaluates multiple facets of effective network flow generation. This benchmark could be significantly enriched by the methodologies and criteria established in the broader field of tabular data generation.

3.2.3.3 Inadequate Evaluation of Temporal Dependencies

The evaluation of temporal dependencies in generated network traffic is still in its early stages. Current methods often focus on numerical features, neglecting the temporal relationships in categorical features, which are crucial for accurately simulating network behavior. Additionally, averaging autocorrelation differences across all time lags can obscure necessary dependencies at specific intervals, leading to evaluations that may not fully reflect the proper temporal structure of the data. More sophisticated techniques are needed to assess whether generated traffic accurately preserves the temporal dependencies between numerical and categorical network traffic attributes.

3.3 Summary

In our pursuit of generating network flows for evaluating NIDS performance, several promising avenues for advancing the state of the art are emerging:

- Develop methods that challenge the dominance of GAN by exploring the potential of non-neural network statistical approaches, such as Bayesian networks, to address the problem. This direction is supported by recent successes of classical statistical methods in similar contexts, such as medical data generation [83].
- Introduce a generative method capable of producing time-dependent network flows without compromising the quality of independent network flows, addressing the shortcomings observed in approaches like NetShare.
- Establish a standardized framework for evaluating the quality of generated network flow datasets that minimizes assumptions about the dataset's intended use. This approach would help avoid the pitfalls associated with the overreliance on utility metrics.

We deal with those three points in the rest of this thesis.

Chapter 4

Individual Network Flows Generation With Bayesian Networks

4.1 Motivations

There is a pressing need to augment the volume of legitimate network traffic data, and given the privacy concerns and logistical challenges of collecting benign real-world network traffic, synthetic data generation has emerged as a viable solution. As many recent approaches rely on Generative Adversarial Networks (GAN) to generate synthetic network flows (see Section 3.1), we choose to focus on this method. Furthermore, since network flow is a commonly used data format in NIDS, we center our efforts on generating synthetic network flows specifically.

Despite their popularity, using GAN to generate network flows brings significant challenges, as detailed in Section 2.4. These challenges include difficulty in modeling dependencies among features, handling the mix of categorical and numerical data, and dealing with sparse distributions in high-dimensional spaces. These issues complicate the generation process, often resulting in less realistic synthetic network flows. Additionally, training GAN is resource-intensive and time-consuming, requiring substantial computational power, which we define here as a lack of *efficiency*. In this context, efficiency refers to minimizing computational power while achieving high-quality output.

Given these challenges, a key question arises: *Does the quality of the traffic generated by GAN justify their high computational cost?* While GAN are known for their ability to capture complex dependencies in data, their performance may not always outweigh the resource costs. This leads us to explore Bayesian Networks as an alternative method that may offer a more qualitative and efficient approach to generating individual network flows. Bayesian Networks, as we argue, provide comparable—or even superior—results while requiring less computational power, potentially offering a better solution than existing GAN-based methods.

4.1.1 Individual Network Flow Generation

Despite the interdependencies between network flows in real-world traffic, generating individual network flows independently is a necessary first step in our approach. This step allows us to ensure that each generated flow is realistic on its own, focusing on capturing the fundamental characteristics of individual flows without the added complexity of temporal dependencies. If we were to generate time-dependent flows directly, we would face the challenge of ensuring both the realism of the individual flows and their temporal relationships at the same time, which could result in errors in both aspects. By first mastering the generation of accurate individual flows, we create a strong foundation that simplifies the incorporation of temporal dependencies later.

Additionally, as highlighted in Section 3.2, GAN that incorporate temporal dependencies, such as NetShare, encounter significant difficulties in modeling the inter-feature dependencies even at the flow level. Therefore, our approach aims to produce more realistic individual network flows with fewer computational requirements, avoiding these issues in the early stages of development. Once we are confident in the quality of the individual flows, we can then extend our work to integrate temporal dependencies in Section 5.4, ensuring that we address both aspects (flow realism and temporal patterns) sequentially and effectively.

4.1.2 Rationale for Using Bayesian Networks for Synthetic Network Flow Generation

Our objective is to explore whether a more efficient and higher-quality alternative to GAN for generating individual network flows exists. As discussed in Section 2.4, this task is similar to generating synthetic tabular data. Bayesian Networks (BN) have demonstrated advantages in similar tabular data generation tasks, particularly in capturing relationships between features and learning marginal distributions more accurately, as evidenced in domains like synthetic patient data generation (see Section 3.1). These strengths arise from the ability of BN to model conditional dependencies, which is critical when generating realistic tabular data.

In the context of network flow generation, where certain features may be conditionally dependent on others, this advantage becomes particularly relevant. The ability of BN to model these dependencies offers a potentially effective solution for generating synthetic network flows, a challenge that has been under-explored. However, network flow datasets may contain more numerical variables compared to datasets in other domains like medical data, which can pose difficulties for BN. This presents an opportunity to investigate whether BN can still provide a

robust and accurate solution for generating synthetic network flows despite these challenges.

An additional advantage of Bayesian Networks (BN) is their explainability. BN are inherently interpretable, representing conditional dependencies between features through a directed acyclic graph (DAG), which clarifies how each feature influences or is influenced by others in the dataset. This transparency is especially beneficial in network flow generation, where specific traffic types can be generated or omitted to match network scenarios, supporting the adaptability discussed in Section 1.2.

However, it is also essential to acknowledge the limitations of BN. As mentioned in Section 2.4, these models face challenges with scalability and are less effective when dealing with continuous features or datasets with many features or categories. Despite these obstacles, the potential benefits of BN make them a compelling candidate for this investigation.

4.2 Research Objectives and Contributions

The goal of this research is to determine to what extent Bayesian Networks (BN) can serve as a viable alternative to Generative Adversarial Networks (GAN) for generating individual network flows. Specifically, we aim to explore whether BN can generate high-quality synthetic network flows while offering reduced computational costs compared to GAN.

To address this, we focus on the following key research questions:

- Are BN preferable to GAN for generating high-quality synthetic network flows with lower computational overhead?
- How does the effectiveness of BN versus GAN vary when considering different sets of features?
- Are the conclusions regarding the use of BN versus GAN consistent across different datasets?

This chapter introduces two key contributions. First, it presents a method that leverages Bayesian Networks to generate individual network flows, addressing challenges such as reducing the cardinality of categorical features and discretizing numerical features. Second, it introduces a comprehensive benchmark designed to evaluate the quality of the generated individual network flows. This benchmark draws on recent advances in the evaluation of tabular data generation to provide a thorough assessment of synthetic network flows across multiple criteria.

4.3 Bayesian Networks for Network Flow Generation

In this section, we present the first contribution of this chapter: a solution to implement Bayesian Networks (BN) for generating individual network flows.

Network flow data is a specific subcategory of tabular data. BN can face challenges in modeling such data, especially when it presents a mixture of numerical features and categorical features with high cardinality. As shown in Table 4.4, our datasets contain a combination of both numerical features, like bytes, and categorical features, like IP addresses, that can span up to 2^{32} distinct values. Therefore, to effectively implement BN for network flow generation, we need to address these issues.

4.3.1 Addressing Challenges for BN on Network Flows

4.3.1.1 Reducing the Cardinality of Discrete Features

In a BN, the size of a Conditional Probability Table (CPT) for a node grows *polynomially* with the cardinality d, following the formula $O(d^{k+1})$, where k is the number of parent nodes [62]. This complexity becomes particularly relevant when considering features in network flows with potentially high cardinality, such as IP Addresses and Ports.

IP Addresses In network flow data, IP addresses can be categorized as either public or private. Public IP addresses typically represent external hosts and are often anonymized in intrusion detection system research [6], [8], [102]. Since their specific values are less meaningful in this context, we can reduce the cardinality by treating all public IPs as a single category. The actual value of the public IP address is less significant than the fact that it belongs to an external host. This approach maintains sufficient detail for evaluating intrusion detection systems while simplifying the model, making it more manageable for BN.

Ports Like public IP addresses, ephemeral ports—temporary ports assigned by operating systems—hold limited informational value. Their actual value is less informative than the fact that they are ephemeral. Therefore, similar to how public IP addresses are treated, we group all ephemeral port values into a single category to reduce cardinality. However, certain ports are non-ephemeral and associated with well-known services (e.g., HTTP on port 80).

Relevance to Intrusion Detection These reductions in cardinality do not negatively impact the subsequent task of intrusion detection. Ports and IP addresses, while useful as identifiers, are

typically not directly analyzed by Network Intrusion Detection Systems (NIDS). What matters more is the recognition of targeted services and the origin of an attack (internal or external). By reducing the cardinality of these features, we maintain model simplicity without compromising the accuracy or effectiveness of NIDS evaluation.

4.3.1.2 Discretizing Numerical Features

BN require all variables to be discrete in order to compute their CPTs. However, network flow datasets contain several continuous numerical variables, such as the number of bytes or the average inter-arrival time of packets. A naive approach to discretize these features would be to treat each unique numerical value as a distinct category. However, this would lead to an explosion in complexity, similar to what happens with high-cardinality categorical features.

To address this, we discretize numerical variables into a limited number of categories. We arbitrarily decide that each numerical variable should be discretized into at most 40 categories. This threshold provides a balance between preserving the granularity of the data and managing the complexity of the BN. We employ two strategies for discretizing numerical features:

- **Quantile Discretization**: This method divides the values of a numerical feature into intervals such that each interval contains an equal number of values. This ensures that the categories capture the distribution of the feature.
- **VGM Discretization**: This method fits a Variational Gaussian Mixture Model (VGMM) to the distribution of the numerical feature. The continuous values are then clustered into discrete categories, with each category representing a Gaussian component. This method allows the discretization to reflect the underlying probabilistic structure of the data.

4.3.2 Implementation with the `bnlearn` Python Library

To implement BN for generating network flows, we use the popular `Python` library for BN on tabular data called `bnlearn`[1]. It is important to note that this library is not equivalent to the `R` library that shares the same name[2], but is instead based on the `pgmpy` framework[3].

[1] Taskesen, E. (2020). We are learning Bayesian Networks with the `bnlearn` Python Package. (Version 0.3.22) [Computer software]. https://erdogant.github.io/bnlearn

[2] https://www.bnlearn.com/

[3] https://pgmpy.org/

As discussed in Section 2.3, training a BN involves two steps: learning the structure of the network (the dependencies between variables) and learning the parameters of each node (the CPTs of each feature).

To select the appropriate structure learning algorithm, we test various algorithms available in the `bnlearn` library on the **UGR'16** dataset (see Subsection 4.5.1) and computed their respective BIC scores. Our goal was to represent the dependencies among the dataset's features with the fewest possible parameters.

Given a collection of data points $\Omega = (x_i)_{i \leq n}$, a model θ, and k, the number of parameters in θ, BIC is defined by Eq. 4.1:

$$BIC(\theta \mid \Omega) = -2 \sum_{i=1}^{n} \log(\mathrm{P}(x_i|\theta)) + k \log(n) \tag{4.1}$$

Structure Learning Method	BIC Score (Lower is better)
Naive Bayes[4]	1.94×10^7
Chow-Liu	1.80×10^7
Hill Climbing	1.68×10^7

Table 4.1: Comparison of different structure learning algorithms from Python's `bnlearn` library on UGR'16.

A lower BIC score indicates a model that balances a good fit with minimal parameters, thereby avoiding overfitting as shown in Table 4.1; the Hill-Climbing algorithm provided the best BIC score in this experiment, so we choose this algorithm for our study.

After performing structure learning, we obtain the parents of each variable. We compute the CPT at each node since we only deal with categorical features due to our discretization process. Figure 4.1 shows an example of a BN trained on **UGR'16** using the Hill-Climbing structure learning algorithm.

4.3.2.1 Sampling of new network flows using the learned model

Once the CPT for each variable is learned, we sample from the root node (*td*, which is the flow duration, in Figure 4.1), determine the initial variable's value, and then deduce the others using

[4] Naive Bayes is not technically a structure learning algorithm but a method for constructing a Bayesian Network with a predefined structure where all features are assumed to be conditionally independent given the class variable. For example, in a network flow model, Naive Bayes might assume that features like the number of packets and the flow duration are conditionally independent given the transport protocol (in some short-lived UDP connections, the number of packets and the duration of the flow may be independent as illustration). It is included in this table because it is one of the methods proposed by the `bnlearn` library.

Bayes' rule. After sampling all the variables, we can revert the discretized variables back to their original numerical spaces:

- **Quantile Discretization**: The discretized value represents an interval. To obtain the numerical value back, we sample uniformly from within that interval.
- **VGM Discretization**: Here, the discretized value is the index of a Gaussian kernel, from which we store the parameters (mean and variance). The numerical value is sampled from the normal distribution associated with that kernel.

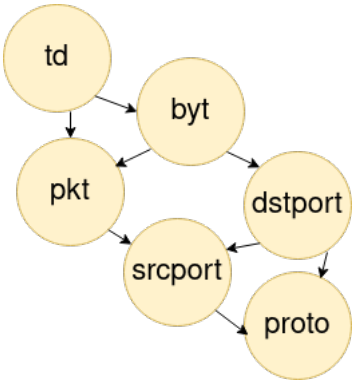

Figure 4.1: Example of a BN structure trained on UGR'16 with Hill-Climbing. Nodes are features; arrows indicate dependencies. The name of the features are explained in Table 4.4

4.4 Evaluation Methodology and Metrics

The goal of this section is to present an evaluation methodology that allows us to compare the quality of the generation of BN and the different GAN models and baselines. Based on what we discuss in Section 3.2, the evaluation of individual network flow generation relies on assessing four criteria:

- **Realism**: A generated network flow should appear to be sampled from the distribution of the training dataset.
- **Diversity**: The distribution of the generated network flows should exhibit the same level of variability as the training dataset.
- **Novelty**: Generated network flows should differ sufficiently from the training ones.

- **Compliance**: Generated network flows should conform to protocol specifications.

We aim to present a methodology that allows for evaluating these criteria for the synthetic network flows generated by each of our methods. To formalize our evaluation benchmark, we outline the main requirements as follows:

- *Exhaustive Evaluation*: The benchmark should comprehensively assess the generated data based on the four predefined criteria: realism, diversity, novelty, and compliance.
- *Individual Criterion Assessment*: Each of the four criteria should undergo an individual assessment to ensure a comprehensive evaluation.
- *Multi-level Evaluation of Realism and Diversity*: Recognizing the unique characteristics of tabular data, Realism and Diversity should be assessed at the marginal distribution, conditional distribution, and joint distribution levels (see Subsection 3.2.1 for an explanation).
- *Data Type Specificity (discrete/continuous)*: To address potential issues related to different data types, the evaluation protocol should incorporate metrics tailored to each data type.

These requirements ensure that our evaluation framework is comprehensive and tailored to assess the quality of synthetic network flows across different dimensions. Table 4.2 provides an overview of the evaluation functions used in our study, outlining the evaluation criteria (realism, diversity, novelty, and compliance), the distributional divergence (marginal, conditional, or joint distribution), and the type of data (categorical or numerical) that these metrics assess. The "Global" line shows the exhaustive nature of our framework, highlighting that it is the first to comprehensively cover all evaluation criteria. This comprehensive framework is in itself a contribution of this chapter.

4.4.1 Comparing Marginal Distributions

To assess both **realism** and **diversity**, we compare the distribution of each feature in the generated network flows with the distribution of each feature in the real network flows. We use the *Jensen-Shannon Divergence* (*JSD*) for discrete attributes of network traffic (such as Protocol) and *Earth Mover's Distance* (*EMD*) for numerical attributes (like Duration or Bytes). These two metrics are widely used for quantitative measurements of distribution divergence [19], [47], [84].

	Criterion				Distribution Type			Data Type	
	Real.	*Div.*	*Nov.*	*Comp.*	*Marg.*	*Cond.*	*Joint*	*Cat.*	*Num.*
JSD	✓	✓			✓			✓	
EMD	✓	✓			✓				✓
CMD	✓					✓		✓	
PCD	✓					✓			✓
Density	✓						✓	✓	✓
Coverage		✓					✓	✓	✓
MD			✓				✓	✓	✓
DKC				✓			✓	✓	✓
Global	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 4.2: Summary of the functions used in our evaluation system. Real.: Realism, Div.: Diversity, Nov.: Novelty, Marg.: Marginal Distribution, Cond.: Conditional Distribution, Joint: Joint Distribution, Cat.: Categorical Data, Num.: Numerical Data

An alternative could be statistical tests like the *Kolmogorov-Smirnov Test* or the χ^2*-Test*. However, unlike these statistical tests, *JSD* and *EMD* do not assume a specific distribution for the features. Moreover, statistical tests provide only qualitative measurements (similar or not similar distributions), whereas *JSD* and *EMD* offer continuous estimations of distribution closeness. Therefore, these functions are more suited to our evaluation of marginal distributions.

Higher *JSD* and *EMD* scores indicate greater divergence between the generated and real network flow data distributions, enabling more accurate comparisons of data quality across different methods. Formal definitions of *JSD* and *EMD* are provided in the Section 3.2.

4.4.2 Comparing Conditional Distributions

Assessing matches in marginal distributions is insufficient; we must also ensure that the generated network flows retain the feature-wise dependencies of the real network flows. For this, popular correlation metrics like the Spearman or Pearson coefficients typically apply to ordered features. However, we need a different approach for unordered categorical features like *Protocol*.

For numerical features, we analyze their correlations using the Pearson correlation coefficient (PCD), as linear correlations are prevalent among features in network flows, such as between the number of packets, total size of packets, and network flow duration. Although the Spearman correlation coefficient can capture more complex correlations, it may not differentiate between linear and nonlinear relationships, making it less suitable for our purposes. We

use the Pearson correlation coefficient to compare inter-feature correlations between real and generated numerical features.

For unordered categorical features, we study the difference between contingency matrices in both the real and generated data. We propose using the *Contingency Matrix Differences* (CMD), which measures the difference between the contingency matrices of a pair of features in synthetic and real datasets. Further details are provided in the State of the Art chapter.

A low CMD or PCD would indicate that the generated data closely mirrors the dependencies and relationships present in the real data. Conversely, a high CMD score or PCD would suggest that the generated network flows fail to accurately replicate the relationships between features accurately, implying less realistic synthetic data.

4.4.3 Comparing Joint Distributions

Metrics based on *PCD* and *CMD* consider only first-order dependencies. However, in network flow data, conditional dependencies often include higher-order information—e.g., the *Number of Bytes* feature may depend on both *Number of Packets* and *Protocol* type. To capture these high-order dependencies, assessing the joint distribution is crucial.

Previous studies often evaluated joint distributions using utility metrics like TSTR or by measuring the performance improvement of a given ML classifier. However, these methods depend heavily on the chosen classification task and ML model architecture. Instead, we use the *Density*/*Coverage* method, directly comparing generated and real manifolds. This approach avoids the bias introduced by specific ML models and is specifically designed to evaluate Realism and Diversity independently, thus meeting the granularity requirement of our benchmark.

As detailed in the Subsection 3.2.1, *Coverage* and *Density* rely on a specified number k of neighbors. According to Naeem et al. [95], the optimal k should be set to 5 when considering datasets with 10,000 samples. In our comparative study, we use this value for k and this sample size.

A high *Coverage* score indicates that the generated data captures a wide variety of samples from the real data distribution, reflecting good diversity. A low *Coverage* score suggests that the generated data fails to cover the real data's range, indicating less diversity. Similarly, a high *Density* score implies that the generated data closely matches the real data's manifold, indicating high realism. In contrast, a low *Density* score suggests that the generated samples are either too sparse or concentrated in areas not representative of the real data, indicating less realistic synthetic samples.

4.4.4 Novelty Evaluation

Inspired by Goncalves et al. [83], we use the *Membership Disclosure* (*MD*) score to measure the Novelty criterion. This score aims to identify synthetic samples that may have been copied from training instances.

To compute the *MD* score, we need a generated set, a training set, and a testing set (the last two being subsets of the real dataset). We calculate the Hamming distance matrix between every pair of generated and real samples. If a synthetic sample has a Hamming distance below a certain threshold r from a real sample, we flag the corresponding real sample as a potential leak. Since we know which real samples belong to the training or testing sets, we can use each r to build a detector of training samples. We then calculate the F1-score of this detector and integrate the F1-score over r. If the generated data includes copies of training samples, the classifier's F_1 integral increases with lower r, indicating potential leaks.

A low *MD* score is desirable to protect patient privacy in medical contexts. However, in network contexts, duplication of network flows (e.g., standard DNS or NTP requests) is expected. Thus, we argue that the *MD* score in synthetic data should be close to that observed in real data. Synthetic data with a low *MD* score may fail to capture the inherent duplication in network data.

Moreover, this score relies on hamming distance and is, therefore, mainly adapted for categorical features. In order to evaluate the Novelty of our generated dataset, we first discretize both our real and generated features using the quantile strategy of Subsection 4.3.1.

4.4.5 Compliance Evaluation

To evaluate the Compliance of the generated network flows, we adapt the *Domain Knowledge Check* (*DKC*) proposed by Ring et al. [16] to our context by customizing the tests to our dataset. *DKC* consists of a set of tests that the generated network flows must pass, ensuring they adhere to standard network rules (e.g., flags only on TCP network flows, etc.). Table 4.3 presents the specific tests used on the different feature sets described in Subsection 4.5.1.

It is important to note that the set of tests is feature-dependent and should be customized for each generated feature set. For example, the rule **"If one of the ports is 53, then the Protocol is UDP"** is applied in all feature sets (CICShortFeatureSet, CICLongFeatureSet, and UGR) because port 53 is used by the DNS service, which operates over the UDP protocol. This rule ensures that flows with DNS traffic adhere to the correct protocol assignment. In contrast, the rule **"If one IP Address is public, then Destination Port is not 137/138"** is not applied in the UGR dataset because the UGR dataset does not have IP addresses. For further details on the nature

of the datasets used, refer to Subsection 4.5.1.

Likewise, as network technologies evolve, some of these tests will likely become outdated. For example, while traditional tests might flag HTTP over UDP as invalid, modern protocols like HTTP/3 over QUIC do indeed operate over UDP. Therefore, future datasets may need updated compliance tests to accommodate new networking standards.

For rules 1, 2, 3, 4, and 6, we borrow them directly from the DKC version of Ring et al. [16]. Rule 5 is explained by the fact that DNS is a service for discovering public IPs linked to specific domain names, making it unlikely to originate from outside our internet network. In CIC-IDS, external requests are handled by a specific external server with a public IP [8], which is also the rationale behind rule 7. Rule 8 is justified by the unidirectional nature of our datasets (see Subsection 4.5.1), while rules 9 to 11 result from the definition of the features.

4.5 Experimental Setup

This section presents the experimental setup we used to address the research questions outlined in Subsection 4.2.

4.5.1 Datasets for Training and Evaluation

To compare the performance of BN and GAN in generating synthetic network flows, we need to use a dataset to train both models. This enables the models to learn how to represent network flows effectively. Our research questions also require us to investigate the impact of the training dataset on both BN and GAN. Therefore, it is essential to select multiple datasets for training.

A survey by Ring et al. [45] categorizes network datasets into three types:

- *Real* traffic captured in a production network. environment
- *Emulated* traffic captured within a testbed or an emulated network environment (that we call *simulated* in this thesis).
- *Synthetic* traffic that was created artificially.

Since we aim to generate synthetic network flows ourselves, we are not interested in synthetic datasets. Therefore, we should compare GAN and BN using both real and emulated datasets to examine how the type of training traffic influences our observations. Additionally, since we focus on generating network flow datasets, we must use datasets specifically containing network flow data.

#	Rule	Features	Experiments		
			CICShortFeatureSet	**CICLongFeatureSet**	**UGR**
1	If the flow has flags, then the Protocol is TCP	Flags, Protocol	✓	✓	
2	At least one IP Address of the flow must be private	Src IP Addr, Dst IP Addr	✓	✓	
3	If one of the ports is 80, 443, or 8080, then the Protocol is TCP	Dst Port, Src Port, Protocol	✓	✓	✓
4	If one of the ports is 53, then the Protocol is UDP	Dst Port, Src Port, Protocol	✓	✓	✓
5	If Source Port is 53, then the Destination IP Address is private	Dst IP Addr, Source Port	✓	✓	
6	If one IP Address is public, then Destination Port is not 137/138	Src IP Addr, Dst IP Addr, Dst Port	✓	✓	
7	If Destination IP Address is public, then Source Port is not 80/443/8080	Src Port, Dst IP Addr	✓	✓	
8	If one port is ephemeral, then the other is an application port	Src Port, Dst Port	✓	✓	✓
9	UDP or TCP flows are not empty	Protocol, Bytes	✓	✓	✓
10	An ICMP flow has 0 bytes	Protocol, Bytes	✓	✓	
11	If the number of Packets is greater than 1, then Duration is greater than 0	Packets, Duration	✓	✓	✓
12	The Duration is not greater than the sum of the inter-arrival times	Packets, IATs, Duration		✓	

Table 4.3: Sample list of tests carried out by the metric **DKC**, with the network rules they assess and the experiments they are associated with.

Moreover, as highlighted by Sarhan et al. [103], different network flow datasets might have different sets of features (see also Section 2.1). This is why one of our research questions investigates the impact of the feature set on our comparison. Thus, at least two feature sets should be investigated. That brings us to at least three datasets, where we should compare one pair of datasets with different types of traffic and another with different feature sets. We propose that the datasets with varying feature sets should be of the same traffic type. This approach allows us to study the impact of these two variables—traffic type and feature set—independently.

Here, we describe the datasets used to train and evaluate BN and GAN. These include the simulated CICIDS 2017 dataset, used to explore the impact of different feature sets, and the real UGR 16 dataset, selected to investigate how real-world traffic influences model performance.

4.5.1.1 Simulated Dataset: CICIDS 2017

For the simulated dataset, we select CICIDS 2017 [8], based on the survey conducted by Ring et al. [45]. Our selection criteria include a simulated dataset containing more than 24 hours of traffic, created from 2017 onwards, and containing pcap files, allowing us to use a custom feature extractor to construct our network flows. To the best of our knowledge, CICIDS 2017 is the only dataset matching these prerequisites.

From CICIDS 2017, we consider the pcap files and use a custom feature extraction tool[5] to create two different feature sets:

- **CICSmallFeatureSet**: This dataset contains benign network flows from five days of traffic simulated in a testbed environment of 12 computers. 11 features describe the network flows.
- **CICLongFeatureSet**: This dataset contains network flows from the same traffic as the CICSmallFeatureSet dataset but with 30 features instead of 11.

These two datasets allow us to compare the impact of the number of features in the training dataset on the results of comparing GAN and BN. The features of the different sets are listed in Table 4.4.

It is worth noting that while CICIDS 2017 initially consisted of bidirectional flows, it allows for the reconstruction of unidirectional flows (See Section 2.1 for the difference between unidirectional and bidirectional flow). Since both NetShare and E-WGAN-GP— the models selected

[5]While it is possible to use the network flows provided directly by the authors of the dataset [8], several studies have identified issues with these flows [104], [105]. Consequently, we opt to use one of the tools proposed by Lanvin et al. [105] in their corrective study. This tool is available at `https://gitlab.inria.fr/mlanvin/crisis2022`.

for assessment (see Subsection 4.5.3)—were initially designed for unidirectional flows, we opt to use unidirectional flows for both **CICSmallFeatureSet** and **CICLongFeatureSet**.

4.5.1.2 Real Dataset: UGR 16

We choose UGR 16 [102] for the real dataset because one of our baselines, NetShare, has been trained on this dataset. To ensure the comparison is as relevant as possible, we use the same subset of UGR 16 as used by NetShare.

The UGR 16 dataset is a real dataset from a Spanish ISP. The specific subset used by the authors of NetShare is the third week of March 2016 from the UGR dataset, which we also decide to use. In this dataset, the network flows are defined by eight features.

It is important to note the absence of IP addresses as a feature in this dataset. Since this dataset originates from real-world network usage, all IP addresses are anonymized by the dataset's authors. Therefore, we decide not to generate IP addresses in this experiment. We refer to this dataset in the following sections as **UGR**.

For all three datasets, 80% of the network flows serve as training data, while the remaining 20% act as a Reference baseline, representing a perfect generation (see Subsection 4.5.3). This results in approximately 3 million unidirectional flows for **CICSmallFeatureSet** and **CICLongFeatureSet**, and around 1 million unidirectional flows for **UGR**. Table 4.4 presents the different features.

4.5.2 Bayesian Networks

We detail our implementation of Bayesian Networks (BN) on our three datasets, applying the solutions to challenges previously resolved in Subsection 4.3.1.

4.5.2.1 Public IP adresses

All public IP addresses can be grouped into a single category in our dataset to reduce feature cardinality without losing meaningful information. Before doing this, we ensure that the multicast address, commonly represented as "0.0.0.0", is not used elsewhere in the dataset to avoid overlap. After this check, all public IP addresses are assigned the value "0.0.0.0". This approach reflects the fact that the specific value of a public IP address is less informative than simply identifying it as external traffic. By treating all public IPs in this way, we preserve the important distinction between internal and external hosts while significantly reducing the complexity of the model.

Feature	**Type**	**Dataset**		
		CICShortFeatureSet	*CICLongFeatureSet*	*UGR*
Source IP Address	categorical	✓	✓	
Source Port	categorical	✓	✓	✓
Destination IP Address	categorical	✓	✓	
Destination Port	categorical	✓	✓	✓
Protocol	categorical	✓	✓	✓
Timestamp	numerical		✓	✓
Day of the week	categorical	✓		
Hour of the day	numerical	✓		
Duration	numerical	✓	✓	✓
Number of packets	numerical	✓	✓	✓
Number of bytes	numerical	✓	✓	✓
Maximum length of a packet	numerical		✓	
Minimum length of a packet	numerical		✓	
Average length of a packet	numerical		✓	
Std of packets lengths	numerical		✓	
Sum of inter-arrival times	numerical		✓	
Average inter-arrival time	numerical		✓	
Std of inter-arrival times	numerical		✓	
Maximum of the inter-arrival times	numerical		✓	
Minimum of the inter-arrival times	numerical		✓	
Flags inside the flow	categorical	✓		
Number of PUSH flags	numerical		✓	
Number of URGENT flags	numerical		✓	
Number of RESET flags	numerical		✓	
Sum of length of the headers	numerical		✓	
Average Number of Packets per second	numerical		✓	
Average of segment sizes	numerical		✓	
Average of Bytes/Bulk ratios	numerical		✓	
Average of Packets/Bulk ratios	numerical		✓	
Average of Bulk Rates	numerical		✓	
Number of packets inside a Subflow	numerical		✓	
Number of bytes inside a Subflow	numerical		✓	
Number of Bytes of the Init Window	numerical		✓	

Table 4.4: Description of the features of each flow in the three datasets of Subsection 4.5.1

4.5.2.2 Ephemeral Ports

Similar to public IP addresses, all ephemeral port values can be grouped into a single category to reduce feature cardinality without losing meaningful information. To handle variations in ephemeral port ranges across different devices, we classify the 30 most frequent ports as non-ephemeral, treating the less frequent ones as ephemeral. These ephemeral ports are assigned the value "99999", indicating that their specific values are not important, only that they belong to the ephemeral port range.

We base the selection of 30 ports on the minimal number n of the most frequent port values, such that the set of ports up to the n-th covers the majority (more than 50%) of network flows. In UGR'16 and CIC-IDS2017, according to Table 4.5, n is 30. The rationale behind this methodology is that most of the traffic in our datasets is either transported by TCP or UDP—protocols where a typical exchange involves one port being non-ephemeral (usually associated with a well-known service or application) and the other being ephemeral (dynamically assigned by the operating system). By considering the set of non-ephemeral ports as those covering 50% of the traffic, we deduce that ephemeral ports likely handle the remaining traffic. This decision is discussed further in Section 4.7.

4.5.2.3 Two discretization methods

We compare two strategies of discretization of numerical features. For this, we create one BN per strategy :

- **$\mathbf{BN_{bins}}$**: a BN that discretizes numerical features by dividing the values into intervals, with each interval containing an approximately equal number of values.

- **$\mathbf{BN_{GM}}$**: a BN that uses a VGMM to discretize continuous numerical features. By fitting the VGMM to the data, each continuous value is assigned to a Gaussian component.

4.5.3 Competing Methods and Baselines

We select a diverse range of GAN implementations to compare the generation quality between GAN and BN. This approach ensures that our results are robust and not dependent on a single implementation of GAN. We choose to compare BN with three GAN models: E-WGAN-GP [16], NetShare [47], and CTGAN [82].

#	Port Numbers	Occurrence
1	53	24.193%
2	80	12.427%
3	443	9.601%
4	123	0.944%
5	137	0.502%
6	25	0.434%
7	0	0.224%
8	22	0.185%
9	8080	0.163%
10	21	0.147%
11	110	0.144%
12	389	0.104%
13	6000	0.083%
14	88	0.082%
15	445	0.077%
16	3306	0.075%
17	138	0.062%
18	22001	0.059%
19	8000	0.052%
20	22011	0.049%
21	3268	0.047%
22	5060	0.045%
23	143	0.045%
24	161	0.044%
25	8888	0.044%
26	139	0.044%
27	5353	0.043%
28	465	0.040%
29	5210	0.039%
30	64887	0.038%
	Sum	**50.036%**

Table 4.5: Distribution of the 30 most frequent port numbers in CIC-IDS-2017 and UGR'16 datasets.

One missing model in our comparison is STAN [19]. It would have been interesting to include STAN as it was demonstrated to perform better than BN in preserving the marginal distribution of network flow attributes. However, we do not include STAN for two reasons:

- Our comparison is focused on GAN since they are the most commonly used type of model for generating network flows, whereas STAN is an autoregressive model.
- Reproducing STAN's work was impractical due to issues with its GitHub repository. The repository is no longer maintained, and key components needed to run the model are outdated or broken, making it impossible to use the existing code as intended.

Reimplementing their autoregressive solution could be a valuable contribution to future work, particularly given the strength of autoregressive models in preserving temporal dependencies (which is out of the scope of this chapter).

In addition to GAN-based competing methods, we include two baselines for comparison: a Real Set and a Naive Sample. In the following, we describe each competing method and baseline, detailing the models' implementations and their relevance to our evaluation.

4.5.3.1 Competing Methods

E-WGAN-GP We choose this model because it is the best-performing model presented in the work by Ring et al. [16]. This WGAN model uses IP2Vec to generate new network flows in the IP2Vec embedding space and then convert them back into the original space. We use the program provided by the author to train E-WGAN-GP on our datasets, sample new flow representations, and convert them back to network flows. We only need to adapt the code so that our dataset features correspond to the original features in the paper. No parameters are changed in the program.

CTGAN We choose CTGAN [82] because it is a standard solution used for generating tabular data [46], [81], providing insight into the expected performance when applying a generic solution to our specific problem. To use CTGAN, we employ the `sdv` library[6], where we specify which features are categorical and which are numerical. Categorical (discrete) features are one-hot encoded. In contrast, continuous features are represented in a multidimensional space, with each coordinate representing the probability that a value comes from a specific kernel,

[6] https://docs.sdv.dev/sdv

determined by a Variational Gaussian Model. These preprocessing steps are automatically handled before the internal functions of the CTGAN model train the model, using the default parameters for configuration and training.

NetShare We select NetShare [47] as it is the most recent solution, allowing us to compare BN with a more advanced approach. To implement NetShare, we use the GitHub repository provided by the authors[7]. Regarding the parameters, we only change the "max_flow_len" and "epoch" values as recommended in the GitHub documentation. The "max_flow_len" represents the maximum number of flows a sequence can contain in one batch during the training phase. While this parameter does not significantly affect the results, it can severely impact the amount of GPU memory required for training. Thus, we set it to 10,000 for the **CICLongFeatureSet** and 20,000 for the **CICShortFeatureSet**. We also set the number of epochs to 400 for both datasets. For the UGR dataset, we do not train the model or sample from an already trained model; instead, we directly use the generated data provided by the authors in their repository.

4.5.3.2 Baselines

Naive Sampler We include the Naive Sampler baseline to provide a reference for comparing the competing approaches. The Naive Sampler generates new data by independently sampling each feature based on its empirical distribution in the training set. This approach assumes that the training set captures the marginal distribution of each feature well but does not account for dependencies between features, making it the worst-case scenario for generation quality.

Real Set The Real Set baseline consists of 20% of the real dataset that was not used during training. This represents the best possible generation quality, serving as an objective for our models to reach. Additionally, as indicated in Subsection 4.4.4, this baseline serves as a reference for evaluating the Membership Disclosure (MD) value of our generated dataset.

4.6 Results of our experiments

In this section, we present the findings from our experiments that are designed to answer the key research questions posed in Subsection 4.2:

[7] https://github.com/netsharecmu/NetShare

- *Are BN preferable to GAN for generating high-quality synthetic network flows while reducing computational costs?* Subsection 4.6.1 evaluates the quality of traffic generated by BN versus GAN on **CICSmallFeatureSet**.
- *How does this conclusion change when considering different feature sets?* Subsection 4.6.2 investigates whether the results remain consistent on **CICLongFeatureSet**.
- *Does this conclusion persist across different datasets?* Subsection 4.6.3 assesses whether the findings hold when considering the **UGR** dataset.

For each experiment, we compare the models using the evaluation benchmark defined in Subsection 4.4 and present the results according to the four criteria: realism, diversity, novelty, and compliance. In the figures throughout this section, **Real** refers to data from the real set (Baseline 2), **Naive** to data generated by the naive sampler (Baseline 1), **BN_bins** represents the BN with quantile discretization, and **BN_GM** represents the BN with VGM discretization.

4.6.1 Experiment on CICSmallFeatureSet

Each metric and model is evaluated on 20 different generated sets for all models, each generated set containing 10,000 network flows (see Subsection 4.4.3 for the explanation of the number of network flows).

4.6.1.1 Realism

Results are shown in Figure 4.2. We can see that the two BN learn the distribution of the discrete features very well, whereas GAN have more difficulty with discrete features, as expected (see Section 2.4). For continuous features, *EMD* shows that BN_{bins} reproduces the marginal distributions better than the GAN (except CTGAN), while BN_{GM} does not manage to learn them, likely due to discretization issues for this dataset.

The *PCD* does not provide much information about the preservation of dependencies among numerical features due to the high variability of the different measurements. Still, the *CMD*, on the other hand, indicates that BN learn the dependencies among categorical features better than GAN. For joint distribution, *Density* shows that BN produce more realistic data than GAN-based methods.

Among the GAN, NetShare underperforms significantly, failing to produce realistic data. Although CTGAN and E-WGAN-GP perform better than NetShare, they do not reach the BN

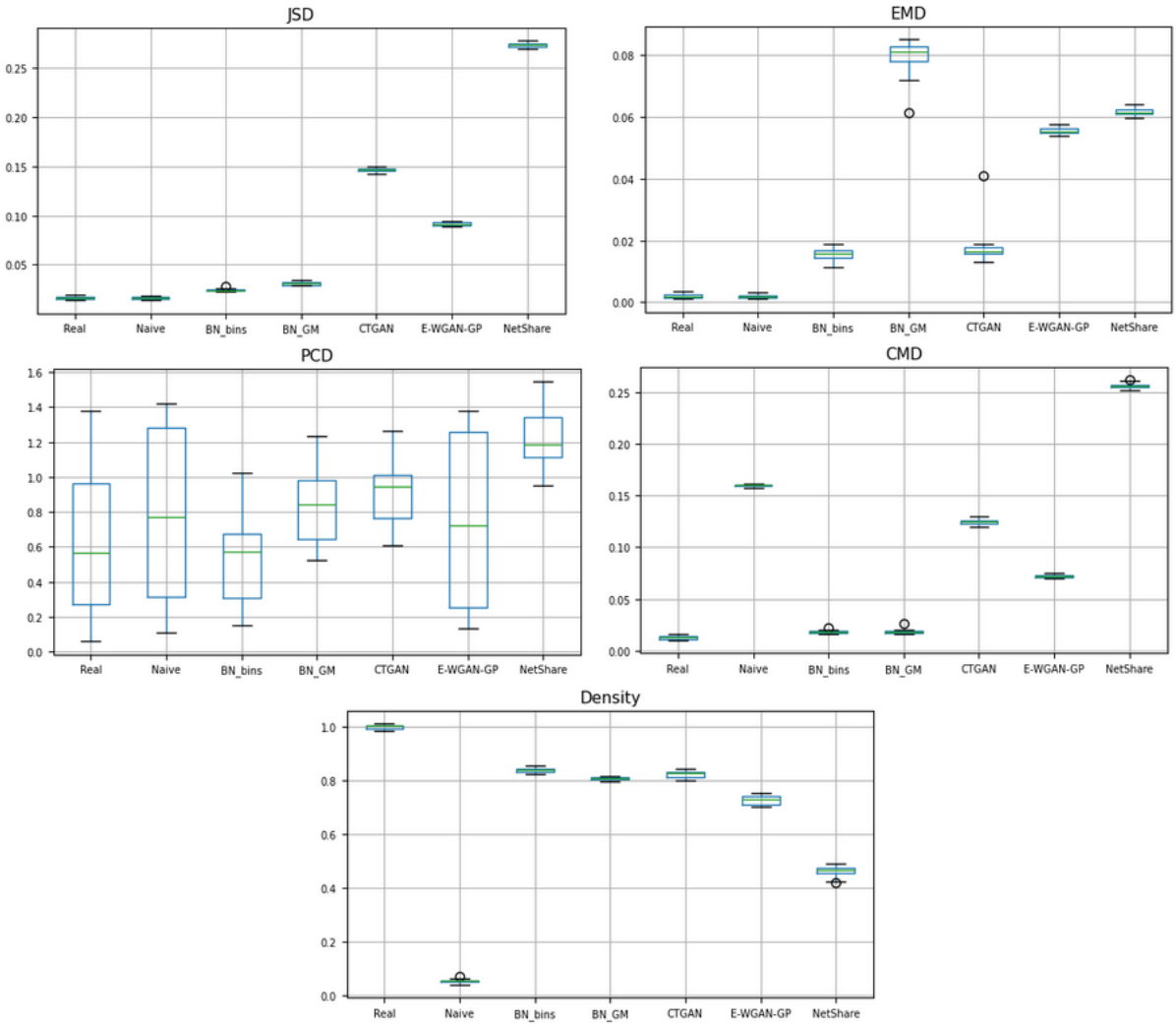

Figure 4.2: JSD/EMD/PCD/CMD/Density of the different models on **CICShortFeatureSet** dataset. Lower is better except for Density, where higher is better.

level. Despite achieving similar *Density* levels, their poorer performance in *JSD* and *CMD* indicates a potential issue with Diversity in the data generated by these models.

4.6.1.2 Diversity

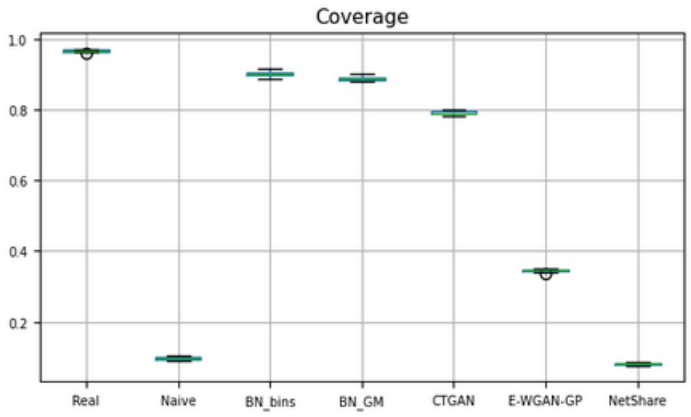

Figure 4.3: Coverage of the different models on **CICSmallFeatureSet**. Higher is better.

As seen in Figure 4.3, BN cover the entire training distribution while CTGAN and E-WGAN-GP underperform in terms of Coverage. This supports the idea that their poorer performance in JSD and EMD is not due to unrealistically generated data but rather to a lack of diversity (JSD and EMD evaluate both realism and diversity). This seems to indicate a phenomenon known as mode collapse, where a GAN fails to cover all the variability in the data. In addition to producing unrealistic data, NetShare also was unable to cover the training data adequately.

4.6.1.3 Novelty

Figure 4.4 shows the Membership Disclosure score of the different models. This score represents the ease with which training samples can be recovered from the generated data. In a typical tabular data generation setup, a lower score would be preferred, but, as explained in Subsection 4.4.4, duplication of network flows is common. Therefore, we argue that MD in synthetic data should be close to MD observed in real data.

For example, NetShare has a low novelty score, meaning it is challenging to recover training samples from its generated data. While this could be seen as positive in some contexts, it likely indicates that the data generated by NetShare is not realistic enough. BN do not appear to leak more data than the test set.

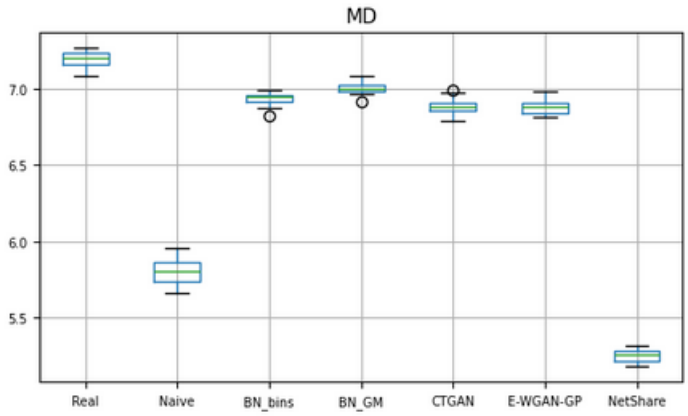

Figure 4.4: Membership Disclosure of the different models on **CICSmallFeatureSet**. Closer to the Real value is better.

4.6.1.4 Compliance

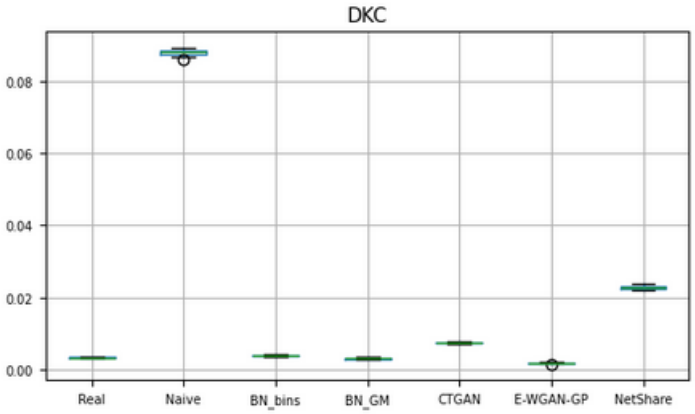

Figure 4.5: Domain Knowledge Check of the different models on **CICSmallFeatureSet**. This score represents the average number of rules one generated network flow breaks. Lower is better.

DKC measures compliance in Figure 4.5. Except for NetShare, all generative models have relatively low DKC scores, indicating that they produce data that is compliant with network specifications. It is also noteworthy that the DKC of the real data is not zero, implying that some abnormal network flows are present in the initial dataset. For example, around 4% of the initial dataset is composed of TCP flows of length 0 bytes, contradicting rule number 9 of Table 4.3. Similarly, some NetBios flows are directed outside the local network, contradicting rule number 6 (around 0.3%). Those flows may be an artifact of the feature extrusion tool we used.

4.6.1.5 Overall

Table 4.6 presents the results of the different metrics for each system. The number in each cell represents the median value for all observations. For every metric, we rank the median of every model apart from the Real set, with 6 to 1, with one being the best. We also average this rank overall metrics to provide a global rank value for every model. This value provides a rough indication of which model may generate the most realistic synthetic data, but it should not be taken as an absolute performance measure.

The analysis of **CICSmallFeatureSet** data reveals that BN-based methods outperform the GAN models in terms of the four criteria of generation quality, with NetShare underperforming even compared to the Naive Sampler. Despite the occasional superiority of GAN models in specific metrics like *EMD*, BN demonstrate consistently superior performance across a range of scoring functions, securing a favorable average ranking across all metrics.

The struggle of GAN-based methods to maintain strong performance in metrics that evaluate the preservation of variable dependencies, such as *CMD* and *PCD*, underscores the ongoing challenge these models face in accurately capturing correlations among variables—a challenge BN-based approaches seems better equipped to handle. Therefore, we can conclude that BN produce better-quality network flows than GAN. In the following experiments, we examine whether this conclusion holds when changing the feature set or the dataset.

	Real	Naive	BN_{bins}	BN_{GM}	CTGAN	E-WGAN-GP	NetShare
JSD	**0.017**	**0.015**	**0.026**	0.031	0.149	0.091	*0.273*
EMD	**0.002**	**0.002**	**0.015**	*0.078*	0.016	0.055	0.063
CMD	**0.013**	0.160	**0.021**	**0.020**	0.128	0.071	*0.256*
PCD	**0.483**	1.155	**0.609**	**0.783**	0.819	*1.133*	1.115
Density	**1.003**	*0.058*	**0.843**	0.813	**0.816**	0.710	0.478
Coverage	**0.970**	0.100	**0.902**	**0.892**	0.797	0.347	*0.084*
MD	**7.194**	5.846	6.909	**7.002**	**6.929**	6.883	*5.275*
DKC	**0.003**	*0.089*	0.004	**0.003**	0.008	**0.002**	0.023
Global Rank	-	4.25	**1.875**	**2.5**	3.25	3.875	*4.75*

Table 4.6: Comparison between 7 traffic generation methods on CICSmallFeatureSet using all the quality metrics. Value is the median of the measurement. The Real column serves as a standard. For each metric: *Red* indicates the worst model, **Orange** the second-best model, and **Green** the best model. (↑): Higher is better, (↓): Lower is better, (=): Closest to the real data is better. The last line gives the average rank given by all metrics to each model and is here just an indication of overall performance.

4.6.2 Experiment on CICLongFeatureSet

This experiment aims to illustrate how the data generation methods behave when more network features are present in the training dataset. **CICLongFeatureSet** includes 20 additional numerical features compared to **CICSmallFeatureSet** (see Subsection 4.5.1). This increase is expected to challenge the data generation methods, particularly BN, due to the need for the discretization of numerical features. In contrast, GAN are inherently designed to handle numerical data.

4.6.2.1 Realism

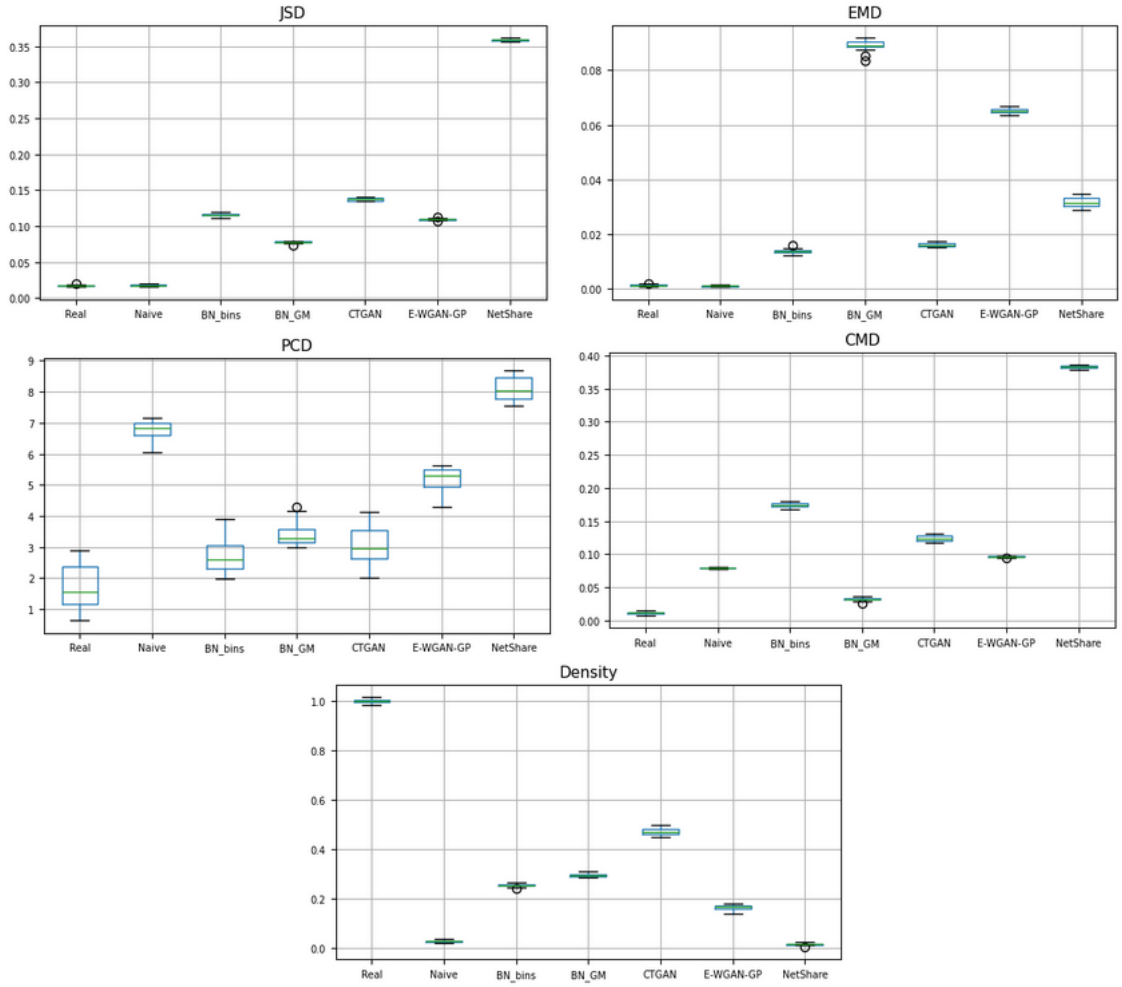

Figure 4.6: JSD/EMD/PCD/CMD/Density of the different models on **CICLongFeatureSet** dataset. Lower is better except for Density, where Higher is better.

The results are presented in Figure 4.6. The *Density* score shows that CTGAN effectively learns the joint distribution of the training data points. When examining *PCD*, it appears that BN, E-WGAN-GP, and CTGAN learn the correlation between numerical features quite well. However, for categorical features, as we can see with *CMD*, they do not perform better than independent sampling (except for BN_{GM}). Regarding marginal distributions, *JSD* indicates that BN methods, E-WGAN-GP, and CTGAN successfully learned the marginal distributions, but *EMD* reveals that BN_{GM} struggles with numerical features, likely due to discretization issues. Overall, CTGAN appears to produce the most realistic samples, as it remains the most consistent across the different metrics.

4.6.2.2 Diversity

Figure 4.7 shows that the Coverage of both BN and CTGAN is superior to the other models but is far from covering the entire real distribution. NetShare performs worse than the Naive Sampler, with E-WGAN-GP only slightly outperforming it.

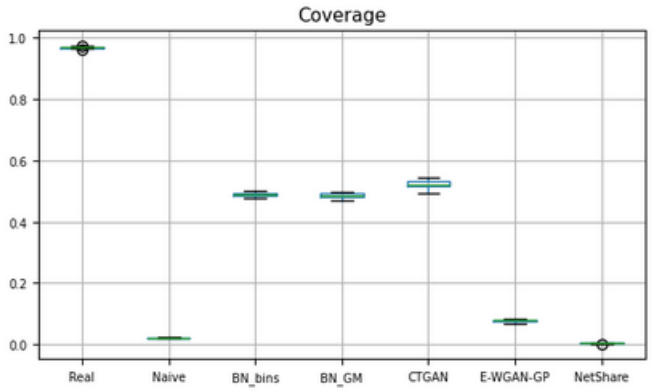

Figure 4.7: Coverage of the different models on **CICLongFeatureSet**. Higher is better.

4.6.2.3 Novelty

As shown in Figure 4.8, the models are further from the real set compared to the first experiment (Figure 4.4). This increased distance is likely due to the additional features introducing more variability into the generation process. E-WGAN-GP is the model with the *MD* closest to the reference.

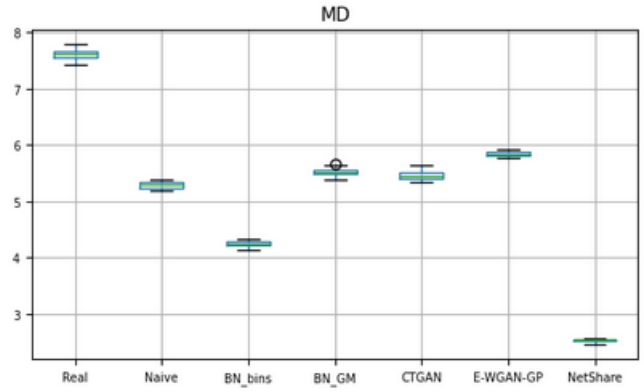

Figure 4.8: Membership Disclosure of the different models on **CICLongFeatureSet**. Closer to the Real set is better.

4.6.2.4 Compliance

BN_{GM} and CTGAN are the two models that produce the lowest DKC scores according to Figure 4.9. It is worth noting that while NetShare did not generate many errors, it also did not produce realistic traffic. This illustrates that compliance and realism, while related, evaluate different properties of the generated traffic.

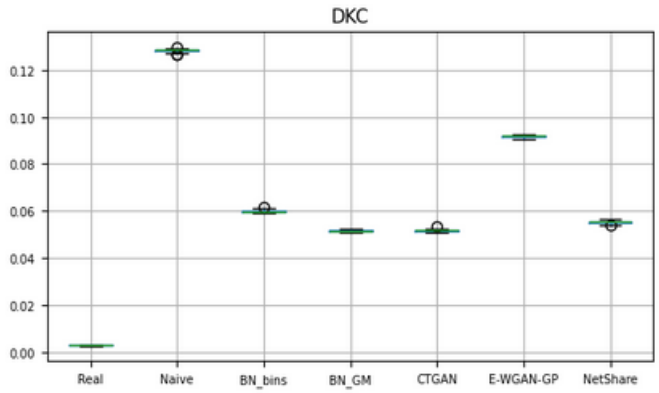

Figure 4.9: Domain Knowledge Check of the different models on **CICLongFeatureSet**. This score represents the average number of rules one generated network flow breaks. Lower is better.

4.6.2.5 Overall

The results on **CICLongFeatureSet** are more contested. While our two BN remain effective at generating data with many features per network flow, Table 4.7 shows that CTGAN matches

our best BN model, BN_{GM}, on several metrics. A case could be made for choosing CTGAN over BN_{GM} for this feature set, particularly when considering the preservation of marginal distributions of numerical features as measured by *EMD*. While BN still outperform E-WGAN-GP and NetShare, certain GAN may perform better when dealing with many features, especially in a feature set focused more on numerical features. Therefore, the response to the second research question is more nuanced than the first. The comparison between GAN and BN appears to depend on the specific feature set under consideration, and the superiority of BN does not hold when we reach a certain number of features in the training set.

	Real	Naive	BN_{bins}	BN_{GM}	CTGAN	E-WGAN-GP	NetShare
JSD	**0.017**	**0.018**	0.116	**0.078**	0.138	0.110	*0.359*
EMD	**0.001**	**0.001**	**0.014**	*0.089*	0.016	0.065	0.031
CMD	**0.011**	**0.079**	0.174	**0.032**	0.123	0.097	*0.383*
PCD	**1.564**	6.842	**2.589**	3.287	**2.963**	5.315	*8.047*
Density	**0.999**	0.026	0.252	**0.292**	**0.468**	0.164	*0.013*
Coverage	**0.969**	0.021	**0.487**	0.485	**0.522**	0.077	*0.002*
MD	**7.614**	5.307	4.240	**5.502**	5.453	**5.836**	*2.543*
DKC	**0.003**	*0.128*	0.060	**0.051**	**0.052**	0.092	0.055
Global Rank	-	3.625	3.125	**2.5**	**2.625**	3.5	*5.375*

Table 4.7: Comparison according to all our metrics of 7 data generation methods using CIC-LongFeatureSetDat. The Real column serves as a standard. For each metric: *Red* indicates the worst model, **Orange** the second-best model, and **Green** the best model. (↑): Higher is better, (↓): Lower is better, (=): Closest to the real data is better. The last line gives the average rank given by all metrics to each model and is here just an indication of overall performance.

4.6.3 UGR

The training dataset in the two previous experiments consisted of simulated traffic generated within an emulated testbed. In this experiment, we use a real-world dataset to observe whether the observations made in the first experiment regarding the superiority of BN hold when we switch to a real-world dataset.

4.6.3.1 Realism

The results are displayed in Figure 4.10. *JSD* shows that BN outperform all GAN in modeling the marginal distribution of categorical features. For numerical features, *EMD* shows that NetShare is the top model for reproducing numerical marginal distributions. Regarding categor-

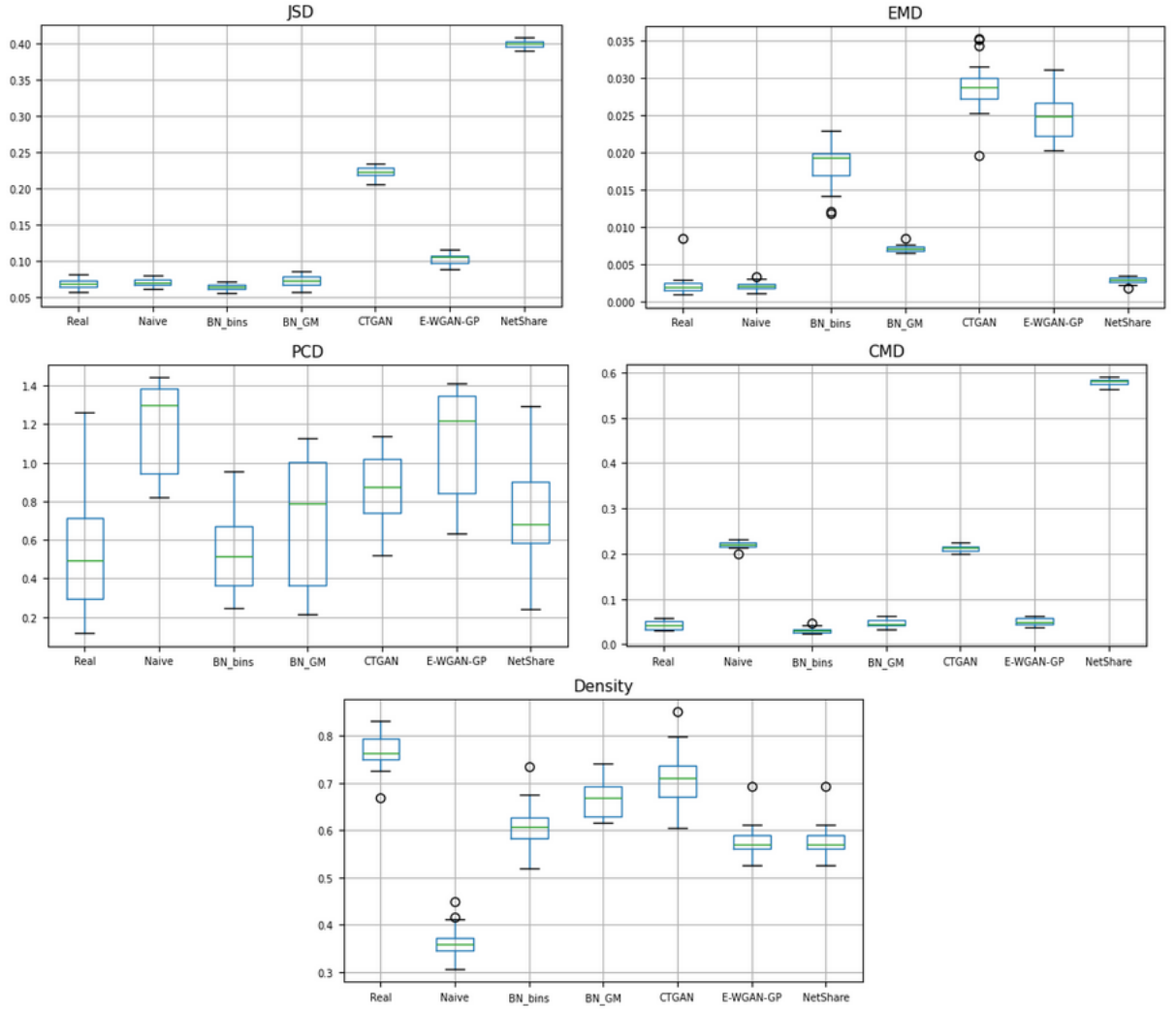

Figure 4.10: JSD/EMD/PCD/CMD/Density of the different models on **UGR** dataset. Lower is better except for Density, where Higher is better.

ical dependencies, *PCD* is inconclusive, while *CMD* shows that BN and E-WGAN-GP best preserve the dependencies among discrete features. Studying the joint distribution with density is also inconclusive due to the high variability of the metric.

4.6.3.2 Diversity

Figure 4.11 shows that the model that best covers the training distribution is BN_{GM}, followed closely by E-WGAN-GP and NetShare. Similar to the first experiment, this may be due to the simpler distribution, which is easier to cover. BN_{bins} has a low *Coverage*, and we can link this to its relatively high *EMD* (see Figure 4.10) to say that it probably does not manage to produce diverse numerical features. This discretization process seems unable to capture that dataset's entire range of numerical values.

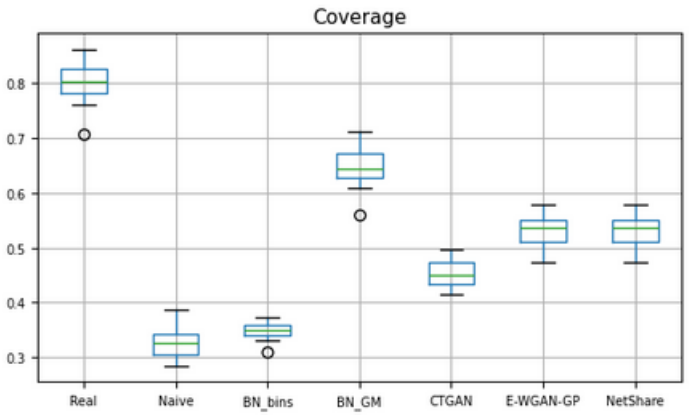

Figure 4.11: Coverage of the different models on **UGR**. Higher is better.

4.6.3.3 Novelty

Due to the simpler distribution (fewer features and smaller cardinality per discrete feature) in the UGR data, it is easier for a data generation model to produce synthetic data close to a training data sample. As a result, the *MD* scores of all models, except NetShare, are close to the real data's *MD*. This experiment shows that the simpler the dataset, the more likely the generated model produces copies of the original data.

4.6.3.4 Compliance

All the data generation methods, except for NetShare, have a good *DKC* score, demonstrating their ability to produce network flows compliant with network protocols. NetShare fails to gen-

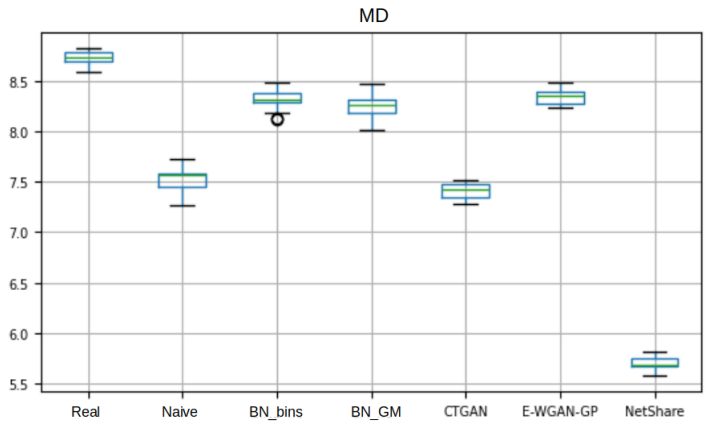

Figure 4.12: Membership Disclosure of the different models on **UGR**. Closer to the Real set is better.

erate valid traffic due to its inability to encode correlations between categorical features.

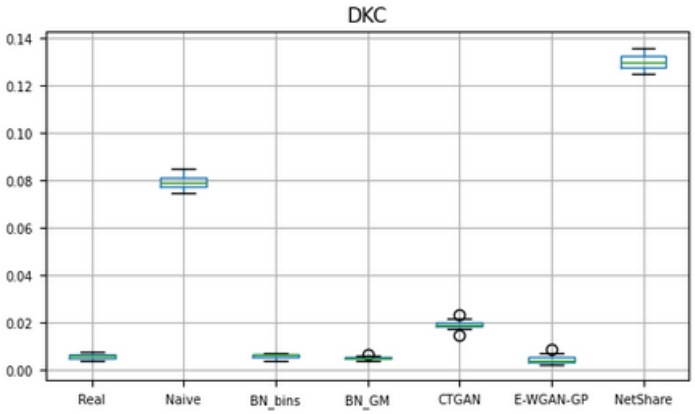

Figure 4.13: Domain Knowledge Check of the different models on **UGR**. This score represents the average number of rules one generated network flow breaks. Lower is better.

4.6.3.5 Overall

Our results reported in Table 4.8 show that BN-based approaches better preserve realism, diversity, and compliance in generated network flows compared to GAN-based methods, mainly when dealing with smaller feature dimensions. Therefore, the superiority of BN is not affected by the origin of the training dataset but rather by the number of features in that training dataset. Worth noting, NetShare showed subpar performance in this experiment, even when assessed using data provided directly by its creators, showing its poor quality in individual flow generation

	Real	Naive	BN_{bins}	BN_{GM}	CTGAN	E-WGAN-GP	NetShare
JSD	**0.069**	0.070	**0.064**	0.073	0.223	0.105	*0.400*
EMD	**0.002**	**0.002**	0.019	0.007	*0.029*	0.025	0.003
CMD	**0.041**	0.221	**0.031**	0.044	0.212	0.049	*0.581*
PCD	**0.495**	*1.296*	**0.515**	0.787	0.876	1.217	0.681
Density	**0.763**	*0.360*	0.607	0.668	**0.711**	0.571	0.571
Coverage	**0.803**	*0.327*	0.349	**0.644**	0.450	0.536	0.536
MD	**8.697**	7.574	8.310	8.302	7.447	**8.363**	*5.685*
DKC	**0.006**	0.079	0.006	0.005	0.019	**0.004**	*0.130*
Global Rank	-	4.375	2.625	**2.25**	4.125	3.125	*4.5*

Table 4.8: Comparison according to all our metrics of 7 data generation methods using UGR Data. The Real column serves as a standard. For each metric: *Red* indicates the worst model, **Orange** the second-best model, and **Green** the best model. (↑): Higher is better, (↓): Lower is better, (=): Closest to the real data is better. The last line gives the average rank given by all metrics to each model and is here just an indication of overall performance.

4.6.4 Computing Cost

Table 4.9 summarizes the computational costs across three key steps: preprocessing, training, and sampling, along with the hardware configurations used in our experiments. In our analysis, BN_{bins} consistently proves to be the most efficient model for generating synthetic samples. Its closest competitor, E-WGAN-GP, benefits from its dense layers, enabling rapid training. However, the reconstruction process required after generation extends its overall generation time, making it slower on the global pipeline. On **CICSmallFeatureSet**, BN_{bins}, BN_{GM}, and E-WGAN-GP exhibit the fastest performance. However, due to the complex IP2Vec embedding reconstruction in E-WGAN-GP and BN_{GM}'s Gaussian Mixture Models (GMMs) training requirement for numerical features, BN_{bins} proves to be the most efficient in terms of time and resources. Conversely, CTGAN and NetShare demand significantly higher computational resources. It should be noted that although Configuration (2) features more powerful hardware, the models running on this setup - CTGAN and NetShare - still require more time to complete the tasks compared to models running on the less powerful hardware of Configuration (1). This highlights that the computational overhead of these models is not solely dependent on hardware but also on their inherent complexity and resource demands. This trend persists in **CICLongFeatureSet**, where the rise in numerical features notably extended preprocessing times for BN_{GM} and training times for all GAN models.

Model	Duration									Epochs	Hardware
	CICShortFeatureSet			CICLongFeatureSet			UGR				
	Prep.	Train.	Samp.	Prep.	Train.	Samp.	Prep.	Train.	Samp.		
BN_{GM}	00:22	00:36	-	1:48	00:44	-	00:19	00:32	-	-	(1)
BN_{bins}	-	00:39	-	-	00:45	-	-	00:32	-	-	(1)
E-WGAN-GP	-	00:11	00:46	-	00:35	0:55	-	00:20	0:21	100	(1)
CTGAN	-	15:02	-	-	19:30	-	-	13:27	-	300	(2)
NetShare	-	20:42	00:39	-	27:31	00:56	-	≃100:00	-	100	(2)

Table 4.9: Computing costs of the three experiments. The format is hours: minutes. Prep.: Preprocessing of the data, Train.: Training of the model, Samp.: Sampling from the model. Hardware configurations: (1) Laptop CPU / 32 GB RAM; (2) A40 GPU / 48 GB VRAM. The training of NetShare on UGR is the order of magnitude given by the authors in their paper. A cell with "-" denotes that either this step does not occur or is so short that we consider its time negligible.

4.6.5 Global Observation

In the experiment on **CICShortFeatureSet**, we observe that BN outperform GAN in generating network flows, delivering better overall quality results at a fraction of the computational cost. However, we aim to determine whether this conclusion holds when the number of features increases.

When the number of features is increased in the **CICLongFeatureSet**, we find that CTGAN performs better, indicating that the initial advantage of BN diminishes in more feature-rich environments. Still, BN remain an attractive option due to their faster training and generation times, while maintaining good overall quality.

We also examine whether these results hold when the source of the training data changes from a simulated testbed to real network traffic, as seen in the **UGR** dataset. Our findings show that the origin of the dataset does not significantly impact the results, and BN continue to perform better overall.

From these experiments, we conclude that the number of features in the training dataset plays a more significant role in determining whether to use BN or GAN than the origin of the dataset. Furthermore, the trade-off between training speed and data generation quality remains a key consideration when selecting a model. Additionally, we note the poor performance of NetShare, even when directly using the artifacts provided by the authors. This highlights the importance of first ensuring that the quality of individual network flows is high before attempting to model time-dependent flows, a criterion where NetShare falls short.

We also present examples of network flows generated by the different models to illustrate their performance. Tables 4.10, 4.11, and 4.12 show sample network flows generated by CTGAN, NetShare, and BN_{bins}, respectively, all trained on the **CICShortFeatureSet**. This dataset is chosen for its readability in a text format. Among the models, BN_{bins} performs the best for

Bayesian Networks on this dataset, while CTGAN and NetShare represent the best and worst performing GAN-based models, respectively.

Across all three tables, we observe the reduction in cardinality as discussed in Subsection 4.3.1, where ephemeral ports are consistently represented by "99999" and public IP addresses by "0.0.0.0." In Table 4.11, NetShare's poor quality is evident. For example, Flows 3 and 5 display incorrect behavior with non-compliant ICMP/IGMP flows featuring PUSH flags. Additionally, there appears to be an overrepresentation of ICMP traffic.

Conversely, both BN_{bins} and CTGAN produce compliant network flows. However, visual inspection alone is insufficient to differentiate the models clearly, further emphasizing the importance of the quantitative analyses discussed earlier.

#	Day	Time	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flags
1	1	66134	66.186061	TCP	192.168.10.16	99999	0.0.0.0	443	14	809	..P...
2	2	52865	10.413489	TCP	192.168.10.12	99999	0.0.0.0	443	13	463	..P...
3	4	60417	84.725331	TCP	192.168.10.50	99999	192.168.10.3	3268	51	2792	..P...
4	3	52249	0.041221	UDP	192.168.10.3	53	192.168.10.17	99999	2	315	.....
5	0	62180	5.255842	TCP	192.168.10.15	99999	0.0.0.0	443	12	523	..PR..
6	1	70590	89.951853	TCP	0.0.0.0	443	192.168.10.9	99999	36	46130	..PR..
7	3	65249	3.217260	TCP	192.168.10.8	99999	192.168.10.50	99999	24	595	..P.U.

Table 4.10: Example of network flows generated by CTGAN

#	Day	Time	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flags
1	0	42170	2.470354	ICMP	0.0.0.0	0	0.0.0.0	0	13	3694	.....
2	0	42179	4.974768	ICMP	192.168.10.14	0	0.0.0.0	0	13	5821	.....
3	0	42193	3.389125	ICMP	192.168.10.9	0	0.0.0.0	0	19	7226	..P...
4	2	42207	2.618284	ICMP	192.168.10.9	0	0.0.0.0	0	8	693	.....
5	0	42207	6.813809	IGMP	192.168.10.14	0	192.168.10.14	0	22	2397	..P...
6	2	42214	6.028982	TCP	192.168.10.3	99999	0.0.0.0	80	15	4397	.....
7	3	42217	10.140024	TCP	192.168.10.12	99999	0.0.0.0	99999	34	2906	..P...

Table 4.11: Example of network flows generated by NetShare

#	Day	Time	Duration	Proto	Src IP Addr	Src Pt	Dst IP Addr	Dst Pt	Packets	Bytes	Flags
1	4	67736	0.172049	UDP	192.168.10.9	99999	0.0.0.0	80	3	288	.....
2	3	43009	0.215606	TCP	0.0.0.0	443	192.168.10.8	99999	12	6238	..P...
3	3	69139	0.000172	UDP	192.168.10.3	53	192.168.10.9	99999	1	152	.....
4	0	68589	6.212143	TCP	0.0.0.0	443	192.168.10.12	99999	6	1530	..P...
5	1	54651	61.298785	TCP	0.0.0.0	443	192.168.10.14	99999	9	6063	..P...
6	1	45805	114.461342	TCP	192.168.10.15	99999	0.0.0.0	443	23	969	..P...
7	1	50268	0.116868	UDP	192.168.10.3	53	192.168.10.12	99999	2	200	.....

Table 4.12: Example of network flows generated by BN_{bins}

4.7 Limitations of the study: Handling of the Discrete Feature Cardinality

A key limitation of our analysis lies in the reduction of the cardinality for discrete features, which may be unfair to GAN models. While BN struggle with high cardinality in discrete features due to the exponential increase in complexity, GAN are better equipped to handle such features. In GAN, a larger number of categories does not significantly affect computational complexity, unlike with BN. Therefore, by reducing the cardinality of discrete features, we may inadvertently restrict the ability of GAN to leverage their full potential.

In particular, our decision to reduce the number of categories for ports to 30 and treat others as ephemeral introduces limitations. While this approach simplifies the model and makes it manageable for BN, it may unfairly disadvantage GAN, which could otherwise handle the full range of port values, such as the potential 65,535 different port numbers. This reduction limits the variability that GAN can capture, potentially affecting the quality of the generated flows.

Additionally, the choice of 30 for non-ephemeral ports has its own constraints. We based this on the empirical observation that the top 30 ports cover more than 50% of the traffic in our datasets. However, this threshold may not hold across different datasets or environments. Traffic patterns and the role of ports can vary, and some less frequent ports that were classified as ephemeral may still carry important information. Conversely, some of the top 30 ports may not always represent stable, non-ephemeral traffic. The fixed threshold of 30 is sensitive to changes in traffic patterns, and may lead to misclassification, especially in dynamic or evolving network environments.

4.8 Summary

In this chapter, we present two significant contributions. The first is a method that implements BN to generate individual network flows. The second contribution is a comprehensive benchmark designed to evaluate the quality of generated individual network flows thoroughly.

We compared our BN-based generation method against GAN across two unidirectional network flow datasets. In the first experiment using the **CICShortFeatureSet**, we demonstrated that our BN method can surpass state-of-the-art GAN-based methods while requiring only a fraction of the computational resources. However, in the second experiment with the more extensive feature set **CICLongFeatureSet**, the gap between BN and GAN narrowed. In this scenario, one of the GAN models, CTGAN, achieved slightly better performance but at the cost of

significantly longer training and sampling times.

Additionally, we compared generation performance using a training dataset from a different recording setup, **UGR**, to determine if this affected the comparison. The results showed that the origin of the dataset did not impact the superiority of BN, as they consistently outperformed GAN, similar to the results obtained with **CICShortFeatureSet**. Therefore, our findings suggest that the choice between BN and GAN is more influenced by the feature set rather than the dataset's origin.

4.8.1 Potential Improvements

Building on this work, there are several avenues for improvement:

- **Utility Criterion in Benchmark:** One potential enhancement could involve adding a "utility" criterion to our benchmark. Currently, our evaluation focuses on neutrally assessing the generated network flows without considering their final usage. However, since the primary goal is to create network flows for evaluating NIDS, testing our generated network flows within an NIDS might be beneficial to see if they are flagged as anomalies. Implementing this would require carefully selecting the NIDS, with justifications for the choice. Although we did not explore this due to our focus on a model-agnostic evaluation, it could be a valuable extension.

- **Privacy-Preserving Criterion:** Currently, the Novelty criterion in our benchmark focuses on preventing the generative model from copying data from the training set, or at least ensuring it does so at a rate similar to that of the test set. However, as highlighted by [106], privacy concerns still arise when using synthetic data generation, as sensitive information can be inadvertently leaked. While our MD metric evaluates the likelihood of training data duplication, we aim to match the score to the test set (which tends to be relatively high, as seen in Figures 4.4, 4.8 and 4.12). From a privacy standpoint, however, this score should be as low as possible to minimize the risk of sensitive data leakage. Currently, our benchmark does not incorporate a privacy-preserving criterion. However, adding one would be especially important when considering model-based generation as a substitute for simulation in the context of sharing datasets for NIDS evaluation.

- **Study of Discretization Processes:** Another area for improvement could be a more in-depth study of the discretization processes used in BN. In this chapter, we only compare two discretization methods, each with a fixed number of 40 categories. We observe

that BN face challenges with discrete features, particularly affecting the performance of BN_{GM}. Further investigation into the impact of this hyperparameter—finding the optimal number of categories or the best discretization method—remains an open question.

- **Impact of Cardinality Reduction on GAN:** A subsequent question arising from this work is the impact of reducing the cardinality of discrete features on the quality of data generated by BN. This aims to address the limitation discussed in Subsection 4.7. By limiting the cardinality of discrete features, one might argue that we are handicapping GAN-based methods. A worthwhile comparison would be to evaluate BN and GAN with varying numbers of categories to determine the point at which the computational complexity of BN outweighs their benefits. It could be done by increasing the number of ports considered non-ephemeral (30 in our study) or by considering some frequently encountered public IP addresses (such as those associated with services provided by companies like Google or Microsoft) as non-anonymized.
- **Temporal Dependencies:** Lastly, another future direction would be to consider temporal dependencies among network flows and develop a generation method that preserves these dependencies. This aspect is addressed in the next chapter.

Chapter 5

FlowChronicle: Synthetic Network Flow Generation through Pattern Set Mining

5.1 Introduction

Simulations are often constrained by their time-consuming nature and their lack of adaptability, as they require to be entirely run again whenever there are changes to the network environment. This makes them impractical, particularly for dynamic or large-scale setups. Our goal is to develop a more scalable and adaptable alternative.

Section 4.6 shows that Bayesian Networks outperform GAN-based methods in generating individual network flows. However, generating only independent flows does not fully capture the complexity of real network traffic. Network flows can occur in sequences where one event triggers another—such as a DNS request preceding an HTTP request. These interactions are part of real-world traffic; they should be considered when generating synthetic data. Thus, beyond generating individual flows, we aim to develop a method that can also capture these sequential patterns, better reflecting the temporal dependencies found in actual network traffic. This will enable synthetic data generation to compete more effectively with simulations, and this chapter outlines our approach to achieve this.

To reproduce temporal patterns in synthetic traffic, we first explain how to compose patterns that can model such dependencies and encode a network flow dataset with them (Section 5.2). We then explain how to select the best set of patterns to represent a network flow dataset using the Minimum Description Length principle (Section 5.3). Building on this foundation, we introduce *FlowChronicle* (Section 5.4), which not only detects these patterns but also uses them to generate synthetic network traffic by sampling from the identified flows. This work is a collaboration with Joscha Cüppers, a PhD student at CISPA. Joscha Cüppers contributed his expertise by assisting with the implementation and drafting of the pattern language and pattern mining methods described in Sections 5.2 and 5.3, while we focused on developing the generative method and setting up the experiments, as detailed in Sections 5.4

and 5.5.

To evaluate the effectiveness of our approach, we address the following key research questions:

- **Does *FlowChronicle* produce better-quality individual flows than existing methods?** In Subsection 5.6.1, we compare *FlowChronicle* to other generative methods using the benchmark presented in Section 4.4.
- **Does *FlowChronicle* better preserve temporal dependencies?** In Subsection 5.6.2, we evaluate *FlowChronicle*'s ability to maintain temporal dependencies more effectively than other synthetic generation methods. To do so, we develop new metrics specifically design for this task in Subsection 5.5.3
- **Is the traffic generated by *FlowChronicle* better than simulated traffic?** Conjointly to the comparison with other generative models, we also compare the traffic produced by *FlowChronicle* with traffic generated by simulation to assess whether *FlowChronicle* generates more realistic data.
- **Does *FlowChronicle* allow for an adaptable generation?** In Subsection 5.6.3, we explore whether it is possible to modulate the generation of *FlowChronicle* by adding specific network activities to the final traffic.
- **Is *FlowChronicle* more efficient[1]?** In Subsection 5.6.4, we compare the computational cost required to run a simulation with the cost required to generate traffic using *FlowChronicle*

By addressing these questions, we aim to demonstrate that *FlowChronicle* is capable of generating realistic, temporally-aware traffic more efficiently than other generative models and investigate its potential its potential as a replacement for simulation when generating network flow datasets.

5.2 Dataset Encoding

In this section, we present the encoding of a dataset using what we refer to as **patterns** and explain how this approach helps in identifying relationships between multiple network flows.

[1] In Section 4.1, *efficiency* is defined as minimizing computational power while achieving high-quality output than a simulation

5.2.1 Pattern-Based Encoding: An Intuition

As discussed in Section 2.1, a network flow dataset can be viewed as a table, where each row corresponds to a network flow, and each column represents a feature of that flow (e.g., source IP, destination port, transport protocol).

Encoding the raw dataset would traditionally involve encoding every value in the entire table. However, certain combinations of values tend to appear frequently, and it becomes more efficient to encode these recurring combinations with a single token. These recurring combinations are what we define as **patterns**.

The main objective is to discover an encoding of the dataset that is as compact as possible. For example, DNS traffic typically involves UDP as the transport protocol and port 53 as the destination port. Instead of encoding the values "UDP" and "53" repeatedly for each DNS flow, we can assign a single token to represent this common combination, thereby compressing the dataset. The patterns we identify are essentially frequent combinations of feature values found through a comprehensive search of the dataset.

While the previous example illustrates a pattern within a single network flow, patterns can also capture combinations across multiple flows. For instance, an IMAP request to read emails may be followed by HTTP(S) requests to fetch images embedded in the email. In this scenario, we can encode the sequence of IMAP and HTTP(S) requests as a pattern, reducing the encoding size and capturing inter-flow relationships.

5.2.2 Formalizing the Concept of Patterns

In this Subsection, we introduce the formal notation used to define patterns and explain the elements that constitute a pattern.

Let F_i represent each feature of a network flow (e.g., Source IP or Destination Port), and let $\underline{F_i}$ denote the set of possible values for feature F_i. Let n represent the total number of features in a network flow. A network flow can therefore be represented as a tuple of n features, and the domain of all flows is the Cartesian product $\underline{F_1} \times \underline{F_2} \times \cdots \times \underline{F_n}$.

We denote a timestamp associated with each flow as t. Thus, a dataset D is a sequence of timestamped network flows, each represented as (t, f), where f is a tuple consisting of the feature values for a specific flow at time t. Our goal in dataset encoding is to shift from this explicit representation of flows to a representation based on **patterns**, which allows for a more compact description of the dataset.

Partial flows serve as the building blocks of patterns. These are tables with the same

columns as the dataset, and each row corresponds to a flow. In a partial flow table, each cell can be one of three types:

- **Fixed**: The value in the cell is fixed and explicitly provided.
- **Reused**: The value in the cell is reused from another cell in a previous row of the same partial flow. A common example would be a Destination IP reusing the Source IP of an earlier flow.
- **Free**: The value of the cell is undetermined by the partial flow and is left to be inferred by a Bayesian Network (BN), which we describe below.

Partial flows help us capture dependencies both within and between flows. However, to fully describe a dataset using partial flows, two additional elements are required: (i) the timestamps of the flows within the partial flows, and (ii) the values for the Free cells.

To assign values to Free cells, each partial flow is associated with a Bayesian Network (BN), which is used to sample values for the Free cells based on the dataset's distribution. Thus, a **pattern** consists of three key components:

1. A **partial flow table** that defines the relationships between feature values.
2. The **timestamps** associated with the flows in the partial flow.
3. A **Bayesian Network** that models the dependencies between Free cells and assigns values to them.

5.2.3 Using Patterns to Encode the Dataset

A **model** is defined as a set of patterns. For each pattern in the model, the occurrences of that pattern in the dataset are referred to as **windows**. A window is a list of indices corresponding to the rows (flows) in the dataset that match the pattern. Since a pattern may occur multiple times within the dataset, it can have multiple windows. However, each pattern must have at least one window in the dataset. For instance, in Figure 5.1, pattern p is associated with two windows: $(12, 89)$ and $(178, 206)$, while pattern q is associated with the window $(56, 113, 145)$.

In Figure 5.1, we illustrate how patterns encode a dataset. On the left, we see three patterns defined by their partial flows and Bayesian networks. Pattern ϵ has one partial flow, where all cells are Free, marked by β. Pattern p has two partial flows. The identifier A in the Reused cell

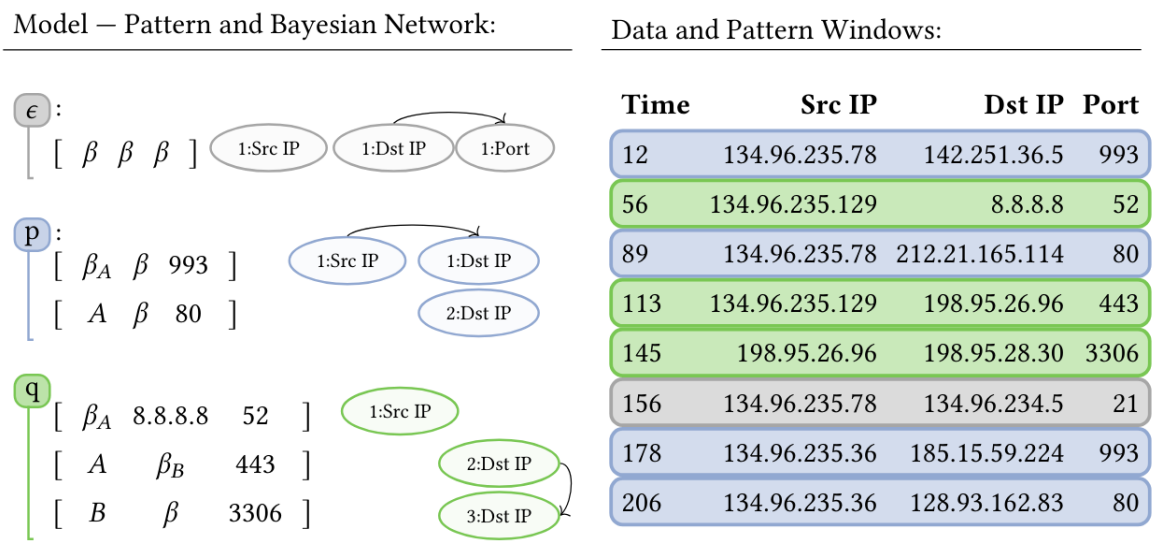

Figure 5.1: Toy example. On the left, we show a model with three patterns; on the right, we show how the patterns cover the dataset.

ensures that the source IP in the second flow is the same as in the first flow's cell β_A. The destination ports are fixed, while the destination IPs remain Free. Finally, pattern q has three partial flows, with Reused cells ensuring that the source IP of the second flow matches the source IP of the first flow and that the source IP of the third flow matches the destination IP of the second flow.

On the right side of Figure 5.1, we observe the occurrences of each pattern in the dataset and deduce the windows for each pattern. Pattern p is associated with two windows: $(12, 89)$ and $(178, 206)$. Pattern q is associated with the window $(56, 113, 145)$. Notice that the fixed values always match the dataset, and the Reused cells are consistent; for example, in each occurrence of pattern p, both source IPs are identical.

To ensure full coverage of the dataset, every flow must be included in a window. However, not all flows stem from sequential activities or recognizable patterns; some flows may represent irregular traffic or "noise". To address these cases, we introduce the concept of an **empty pattern**, which consists of a single row where all cells are Free. This allows us to handle flows that do not fit into any identified patterns by generating their values through the Bayesian Network. In Figure 5.1, the pattern ϵ represents such an empty pattern. It ensures that even flows without clear relationships are still accounted for within the model. This guarantees that every flow in the dataset is assigned to at least one window, thus fully encoding the dataset using patterns. By using this approach, we can go from a description of a dataset in terms of timestamps

and network flows to a description in terms of patterns and windows.

5.3 Minimum Description Length

Now that we can describe the dataset in terms of patterns and windows, our next task is to choose the best model (set of patterns) among various ways to encode the dataset. To determine the best model, we employ the Minimum Description Length (MDL) principle, which asserts that the best model is the one that compresses the dataset most effectively.

The MDL principle [107] is a model selection criterion grounded in Occam's razor, which suggests that the simplest explanation is often the best. Formally, MDL is an approximation of Kolmogorov complexity [108]. For a given class of models $\mathcal{M}$, MDL identifies the best model $M \in \mathcal{M}$ as the one that minimizes the total number of bits required to describe both the model and the data given the model:

$$L(D, M) = L(M) + L(D|M)$$

where $L(M)$ is the length of the model M in bits, and $L(D|M)$ is the length of the data D given the model M. In this section, we explain how to compute both $L(M)$ and $L(D|M)$.

5.3.1 Length of a Model

A model M is composed of a set of $|M|$ patterns. To encode the model, we need to encode the number of patterns $|M|$ as well as each individual pattern. Therefore, the total length required to encode the model is:

$$L(M) = L_{\mathbb{N}}(|M|) + \sum_{p \in M} L(p) \quad .$$

5.3.1.1 Encoding the Number of Patterns $|M|$

The number of patterns $|M|$ is encoded using a formula derived from Rissanen's method for encoding integers [109], which is given by:

$$L_{\mathbb{N}}(|M|) = \log^* |M| + \log c_0 \quad ,$$

where $\log^* |M|$ represents the expansion $\log |M| + \log \log |M| + \ldots$[2], and we continue the expansion until the value of the next logarithm becomes negative, at which point the series stops. Rissanen suggests using $c_0 = 2.865064$. This formula allows us to compute the number of bits required to encode the number of patterns in the model.

5.3.1.2 Encoding the Patterns

To encode a pattern p, we need to encode (i) the number of partial flows it contains, (ii) each of these partial flows, and (iii) the Bayesian Network (BN) that models the Free cells within the pattern:

$$L(p) = L_{\mathbb{N}}(|p|) + \sum_{j=1}^{|p|} L(X[j]|p) + L(BN_p) \quad .$$

The number of partial flows $|p|$ is encoded using Rissanen's formula. Next, we detail how the Bayesian Network and the partial flows are encoded.

Encoding the Bayesian Network We encode the Bayesian Network (BN) by first encoding the number of parents c for each node in the network using $\log K$ bits, where K is the maximum number of parents allowed for any node[3]. We then select the parents from the remaining $|B|-1$ possible nodes, where B is the set of Free cells defined by the BN. The encoding length is:

$$L(BN_p) = \sum_{(j,i) \in B_p} \log K + \log \binom{|B| - 1}{c_{j,i}} \quad ,$$

where B_p is the set of Free cells in pattern p.

Encoding the Partial Flows The encoding of partial flows is split into three components:

1. Encoding the Fixed cells.
2. Encoding the Free cells marked for reuse.
3. Encoding reused values from earlier flows, if any.

To encode the Fixed cells, we first encode how many of the n flow features have a fixed value, which requires $\log n$ bits. To select the k fixed cells, we use $\log \binom{n}{k}$ bits. The values themselves are encoded by selecting from the domain of each feature.

[2] All logarithms in this part are base-2

[3] K is a hyperparameter of the training, in our experiment Section 5.5, we use $K = 3$

For the cells marked for reuse, we first encode how many of the remaining $n-k$ cells are marked for reuse, followed by selecting the l cells to mark. Finally, for cells that reuse values, we encode how many of the remaining $n-k-l$ cells reuse values, and we select the m cells to reuse, along with the specific earlier cells to reference. This encoding is formalized as:

$$\begin{aligned} L(X[j]|p) = &\log(n) + \log\binom{n}{k} + \sum_{i \in S_j} \log|\underline{F}_i| \\ &+ \log(n-k) + \log\binom{n-k}{l} + \\ &\mathbb{1}(|\pi(j,p)| > 0)\Big(\log(n-k-l) + \log\binom{n-k-l}{m} + m\log|\pi(j,p)|\Big) \quad , \end{aligned}$$

where S_j is the set of fixed cells in the j^{th} flow, and $\pi(j,p)$ denotes the set of cells available for reuse.

5.3.1.3 Final Formula for the Model Length

Thus, the total length required to encode the model is:

$$\begin{aligned} L(M) = &\log^*|M| + (|M|+1)\cdot\log c_0 \\ &+ \sum_{p \in M}\Bigg(\log^*|p| + \sum_{j=1}^{|p|}\Bigg(\log(n) + \log\binom{n}{k} \\ &+ \sum_{i \in S_j}\log|\underline{F}_i| + \log(n-k) \\ &+ \log\binom{n-k}{l} + \mathbb{1}(|\pi(j,p)|>0)\Big(\log(n-k-l) \\ &+ \log\binom{n-k-l}{m} + m\log|\pi(j,p)|\Big)\Bigg) \\ &+ \sum_{(j,i)\in B_p}\log K + \log\binom{|B|-1}{c_{j,i}}\Bigg) \quad . \end{aligned} \tag{5.1}$$

5.3.2 Length of the Dataset Given the Model

To compute the length of the encoding of the dataset D given the model M, we rely on what is referred to as the *cover* of D using M. The cover is a set of windows (see Section 5.2.3), which describes how the patterns of the model are used to represent the dataset. More precisely, a

cover specifies how frequently each pattern occurs, where these occurrences are located in the dataset, and the values of the Free cells.

5.3.2.1 Illustration of Encoding the Dataset Using a Cover

A cover is a sequence of codes representing how the dataset is encoded with the model. As said above, the cover specifies for each pattern how often it occurs (number of windows), the positions of these occurrences (delays), and the values for the Free cells in each occurrence.

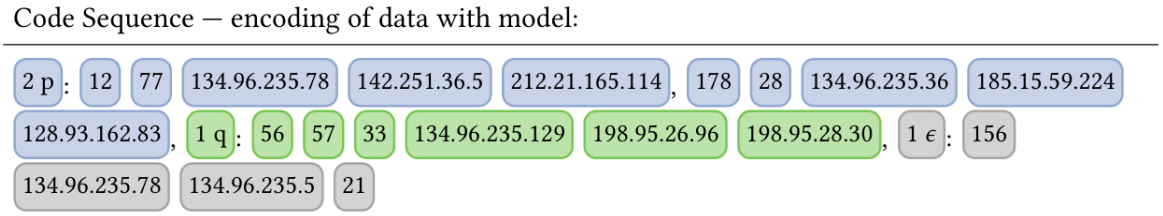

Figure 5.2: Encoding of the data shown in Figure 5.1

In Figure 5.2, we show an example of how the dataset is encoded using the model from Figure 5.1. For example, we first encode the code 2 p, indicating that pattern p occurs twice in the dataset. For each occurrence of p, we then encode the timestamps (12 and 77), which represent the start time and the delay to the next flow (77 + 12 = 89). Next, we encode the values for the Free cells in each flow. This process is repeated for pattern q and the empty pattern ϵ. The result is a full encoding of the dataset in terms of the patterns and their respective windows.

5.3.2.2 Length of the Cover

To compute the length of the cover, we sum the bits required to encode the number of occurrences (windows) for each pattern, the timestamps, and the values of the Free cells, as defined by the Bayesian Network. Formally, the length is:

$$L(D \mid M) = \sum_{p \in M} \left(L_{\mathbb{N}}(|W_p|) + L(W_p) \right) \quad ,$$

where W_p denotes the set of windows (occurrences) for pattern p in the dataset. The length $L(W_p)$ depends on the timestamps of each occurrence and the values of the Free cells, which are encoded based on the probabilities given by the Bayesian Network.

5.3.2.3 Length of the Cover

The total length of the cover is computed by summing the number of bits required to encode how often each pattern occurs (the number of windows) and the bits required to encode the timestamps and Free cells within each window:

$$L(D \mid M) = \sum_{p \in M} \left(L_{\mathbb{N}}(|W_p|) + L(W_p) \right) \quad ,$$

where W_p denotes the set of windows associated with pattern p in the dataset. The length $L(W_p)$ depends on the timestamps and Free cells, which are encoded using the probabilities provided by the Bayesian Network.

5.3.2.4 Final Formula for the Dataset Length

The complete length for encoding the dataset given the model is:

$$\begin{aligned} L(D \mid M) = & |M| \cdot \log c_0 + \sum_{p \in M} \Bigg(\log^*(|W_p|) \\ & + \sum_{i=1}^{|W_p|} \left(L(t_1 \text{ of } w_i) + \sum_{k=2}^{|p|} L(t_k \text{ of } w_i \mid t_{k-1}) \right) \\ & - \log \left(Pr(w_i \mid BN_p, \{w_j \mid j < i\}) \right) \Bigg) \quad . \end{aligned} \tag{5.2}$$

5.4 FlowChronicle: A Model for Network Flow Generation

Having established a framework for encoding datasets with a model and the objective function for mining the best model, we now introduce *FlowChronicle*, a method specifically designed for network flow datasets that mines sequential network flow patterns and generates new data from them. This process involves three key phases: preprocessing the network flows (Subsection 5.4.1), mining and optimizing patterns in the dataset (Subsection 5.4.2), and generating new network flows based on the model (Subsection 5.4.3).

5.4.1 Preprocessing Network Flows

Before training the model, we must preprocess the network flow dataset to facilitate searching for potential patterns. Network flows often contain a mix of numerical and categorical fea-

tures. To enable the identification of patterns, we must first discretize the numerical features in the network flows. We apply a discretization strategy that divides each numerical feature into 40 categories, ensuring that each category contains an equal number of samples, similar to the quantization strategy described in Section 4.3. This discretization strategy was chosen because our dataset closely resembles the CICSmallFeature set (see Section 4.5), and in that context, the bayesian network using a discretization using equal-width bins (BN_{bins}) produced the best results (see Section 4.6).

5.4.2 Pattern Mining

Once the dataset is preprocessed, we aim to identify both inter-feature and inter-flow dependencies, which will allow us to recreate these relationships in synthetic network flows. To achieve this, we apply the Minimum Description Length (MDL) principle, as described in Section 5.3. The process is iterative: at each step, we add patterns to our model to reduce the overall description length. The patterns added at each step are chosen from a set of candidate patterns, which expands as the mining process progresses.

5.4.2.1 Creating Candidate Patterns

Before discussing how we select the best candidate pattern at each step, we need to explain how candidate patterns are generated. As explained in Section 5.2.2, a pattern consists of three main components: a partial flow table, the timestamps for each flow, and a Bayesian Network (BN). At each iteration of the mining process, we generate candidate patterns from existing patterns in the model through the following three options:

1. **Fixed Cell Creation**: A Free cell is converted into a Fixed cell, or a new row of Free cells is added, with one of them transformed into a Fixed cell. It is important to note that source and destination IP addresses cannot be fixed, as we do not want our patterns to depend on specific IP addresses in the dataset.

2. **Pattern Merging**: Two patterns, each with a single partial flow, are merged into a new single-flow pattern if no conflicting Fixed cells exist. For multi-flow patterns, candidates are generated by appending partial flows.

3. **Reused Cell Creation**: A Free cell in a multi-flow pattern is converted into a Reused cell, which references a previously marked cell in an earlier flow.

This iterative process progressively adds patterns to the model. We start with an initial model containing only the *empty pattern*, which covers all the flows in the dataset.

5.4.2.2 Selecting Candidate Patterns

Once candidate patterns are created, we need to select those that optimize the model. At each step, we aim to minimize the **description length** of the dataset, as explained in Section 5.3.

Optimizing the Description Length At each step, we choose the candidate pattern that provides the greatest reduction in the overall description length $L(D, M)$. If a candidate does not reduce the description length, it is discarded from the model.

We maintain two lists: one for candidates that successfully reduce the description length and improve the model, and one for candidates that failed, which are not tested again in future iterations.

Managing the Search Complexity In the early iterations, the number of patterns is small, and thus the number of candidates to evaluate is also low. However, as more patterns are added, the number of candidates increases quadratically (due to the pattern merging operation), which can significantly slow down each iteration.

To address this, we implement a stopping criterion. If t consecutive candidates fail to reduce the description length, we terminate the current iteration and proceed to the next. The value t represents the threshold for consecutive misses, ensuring that the algorithm focuses on the most promising patterns. Additionally, candidates are ranked using a **candidate score**, calculated by multiplying the number of flows a pattern can cover by the number of Fixed or Reused cells in the partial flow.

Additional Restrictions in Candidate Generation In the first phase of candidate generation, we allow the creation of Fixed cells by converting Free cells or adding new rows to partial flows. However, allowing new rows too early often results in uninformative patterns, such as patterns consisting only of TCP flows. To avoid local minima and generate more interpretable models, we restrict the addition of new rows to the second phase of the training process. For the same reason, sometimes a candidate will be an already existing pattern in the model, so we decide to prune redundant patterns in the model every time a new pattern is added.

Summary of the Search Process The search begins with an initial model containing only the empty pattern. Next, the initial set of candidates is created. For each pair of features, and for each combination of values of these two features (except for the source IP and destination IP), we create a pattern with a single flow containing two Fixed cells. The remaining cells are Free and are described by a Bayesian network.

Once the initial set of candidates is generated, we proceed by generating single-row candidates according to the first two options in Section 5.4.2. If a candidate reduces the MDL, it is added to the model; if not, it is discarded. After t consecutive candidates fail to reduce the MDL, we begin generating multi-row candidates and repeat the process. When no further candidates can reduce the MDL, we reach a local minimum, and the final model for the dataset is obtained. The entire search process is summarized in Algorithm 1[4]

Algorithm 1: Main Search Algorithm

Input : set of flows D,
continuous misses threshold t
Output: model M and cover of D

$M \leftarrow \{\text{empty pattern}\}$
$C \leftarrow$ all pairwise combinations of features and values
mode $\leftarrow$ *single-flow*, misses$\leftarrow 0$
while *misses*$< t$ *or mode* $=$ single-flow **do**
  $C \leftarrow C \cup \textsc{BuildCandidates}(M, \text{mode})$
  $c \leftarrow \arg\max_{c \in C} \text{candidate\_score}(c)$
  if $L(D, M) > L(D, M \cup c)$ **then**
    $M \leftarrow M \cup c$
    $M \leftarrow \textsc{Prune}(M)$
    misses$\leftarrow 0$
  else
    misses$\leftarrow$ misses$+1$
  if *misses* $> t$ *and mode* $=$ single-flow **then**
    mode $\leftarrow$ *multi-flow*
    misses $\leftarrow 0$
return M, C

[4]For clarity, the inclusion of failed candidates in a separate list, which ensures they are not tested again, is omitted from the pseudocode.

5.4.3 Generating Synthetic Network Flows

At the end of the training process, we obtain both the model and the cover of the dataset. The cover consists of windows for each pattern, and from this, we can determine the occurrence count of each pattern (including the empty pattern). This allows us to construct a probability distribution over the patterns, from which we sample new patterns to generate a synthetic dataset. Once patterns are sampled, they are assigned timestamps to ensure proper temporal structure in the dataset.

5.4.3.1 Sampling Timestamps

To arrange the patterns in the synthetic dataset, we must sample timestamps for each pattern. Using the windows from the model, we extract the initial timestamps for each pattern in the original dataset. This initial timestamp marks the beginning of the pattern's occurrence.

By gathering the initial timestamps from each window in the training dataset, we can construct a probabilistic model for the timestamps using *Kernel Density Estimation* (KDE).

KDE is a non-parametric way to estimate the probability density function of a random variable. Given a set of observations, KDE constructs a continuous distribution by placing a smooth "kernel" function, in our case Gaussian, over each data point and summing them to form a density curve. This allows us to model complex, multimodal distributions without assuming any particular underlying distribution. In our case, KDE enables us to capture the natural variability in initial timestamps, providing a realistic foundation for sampling new values from our collection of initial timestamps.

Once the initial timestamps are sampled, we must arrange the flows within each pattern. It is possible for patterns to overlap, as demonstrated in Figure 5.1, where patterns p and q intersect.

To generate the timestamps for each flow within a pattern, we examine the interarrival times for the flows in the training dataset. For example, in Figure 5.1, the interarrival times for the second flow in pattern p are $89 - 12 = 77$ and $206 - 178 = 28$. Another KDE is applied to model the distribution of interarrival times for each flow, allowing us to sample new interarrival times for the synthetic dataset.

5.4.3.2 Generating Flows from Patterns

We use the partial flows from each sampled pattern to generate new synthetic network flows. The following steps are used to fill in the cells of the sampled patterns:

- *Fixed* cells retain the values specified in the partial flow.
- *Free* cells are sampled from the Bayesian Network associated with the pattern. Each pattern has its own BN, and we learn the structure and Conditional Probability Tables (CPT) from the training data. The Free cells are then sampled directly from the BN.
- *Reused* cells take the value from the referenced earlier flow within the same partial flow.

5.4.3.3 Postprocessing the Generated Dataset

Because the network flow dataset is discretized in Subsection 5.4.1, the generated synthetic dataset is initially represented by discrete symbols. To convert it back into the original format, we replace each symbol with its corresponding value from the dataset's alphabet. For numerical features, we sample uniformly within the quantile range corresponding to each symbol, thus restoring the numerical data to its original form.

5.5 Experimental Setup

This section outlines the experimental setup designed to address the research questions outlined in Subsection 5.1. We describe the process of training *FlowChronicle* and several baseline methods on a network flow dataset. First, we explain the rationale behind the choice of the dataset, followed by an introduction to the baseline methods used for comparison with *FlowChronicle*. Finally, we describe the methodology for evaluating the performance of these models.

5.5.1 Dataset

The choice of the dataset for our experiment is driven by the necessity of comparing *FlowChronicle* to simulations. When attempting to demonstrate whether *FlowChronicle* generates better traffic than a specific simulation, S, one might consider using a real-world dataset. In this approach, we generate synthetic traffic with *FlowChronicle* and then simulate the real traffic with S for comparison. However, this method only allows for a comparison against that particular simulation rather than simulations in general. The results might reflect how effectively S is configured to replicate real traffic, rather than demonstrating *FlowChronicle*'s potential to serve as a general replacement for simulation.

A more reliable approach is to compare *FlowChronicle* against the best possible scenario for simulation—where the simulation perfectly replicates the real traffic. To achieve this, we can select a dataset composed entirely of simulated traffic and use one time period to train *FlowChronicle*. We then compare the traffic generated by *FlowChronicle* to the traffic from a different time period within the same simulated dataset. In this case, the simulation process that generated the dataset serves as the ideal simulation, as it creates both the training and comparison traffic. If *FlowChronicle* can produce traffic of similar quality to the simulation under these controlled conditions, it would suggest that *FlowChronicle* is capable of surpassing other simulations, which are likely less accurate than this best-case scenario. This method offers a rigorous and fair assessment of *FlowChronicle*'s capability as an alternative to traditional simulation for synthetic traffic generation.

Therefore, to effectively compare *FlowChronicle* with other synthetic traffic generation methods and simulations, we require the reference dataset to meet two key conditions: it should consist of emulated or simulated traffic and span two distinct periods of time. This temporal separation enables us to evaluate whether *FlowChronicle* can achieve the quality of the optimal simulation, thereby outperforming any other simulation.

5.5.1.1 CIDDS-001

The CIDDS-001 (Coburg Intrusion Detection Data Set) is a labeled, flow-based dataset designed for evaluating anomaly-based intrusion detection systems. It was captured over four weeks in an emulated small business environment using OpenStack, containing multiple clients and servers (such as web, email, and backup servers). The dataset includes both benign and malicious traffic. Traffic was captured from various subnets, including internal server-client interactions and external server communications. It contains approximately 30 million network flows.

5.5.1.2 Bidirectional Network Flow

The CIDDS dataset provides unidirectional traffic, but our goal is to study the preservation of meaningful dependencies. To do this effectively, we need to remove the dependencies of responses following requests within the traffic. Therefore, we combine the unidirectional flows into bidirectional flows.

To achieve this, we iterate over the entire dataset and group the unidirectional flows by their 5-tuple: (Source IP Address, Destination IP Address, Source Port, Destination Port, Pro-

tocol). This groups all communications that occur between two hosts over the entire dataset. Naturally, this grouping includes multiple sessions that span several days. We must, therefore, distinguish between different sessions within these communications. This can be achieved for TCP connections when both terminals send their FIN flags to close the connection. However, for UDP traffic, which is connectionless, we need to use a timeout-based approach. We choose a timeout of 120 seconds, which is a standard value used in the literature [37], [75], to define the end of a session for UDP flows.

Ultimately, we combine all unidirectional flows within the same session into a single network flow with 11 features, detailed in Table 5.1. We do not include the Source Port because it is generally randomly sampled within a particular range (see RFC6056 [28]). However, the Source Port is sometimes fixed, such as with protocols like NetBIOS and NTP. In that case, it can be reconstructed using basic network knowledge.

Feature	Description of the feature	Example of value
Date first seen	Timestamp of the first packet of the flow	2017-04-11 00:00:14.387
Proto	Transport protocol	TCP
Src IP Addr	Source IP Address (Client)	192.168.220.255
Dst IP Addr	Destination IP Address (Server)	10425_118*
Dst Pt	Destination Port	443
In Byte	Number of Bytes coming to the client	1165
In Packet	Number of Packets coming to the client	7
Out Byte	Number of Bytes sending from the client	535
Out Packet	Number of Packets sending from the client	5
Flags	Type of flags contained in the flow	.AP.S.
Duration	Duration of the flow	0.369

* Public IP addresses are anonymized in the CIDDS dataset.

Table 5.1: Set of Features and their description in our dataset

5.5.1.3 Benign network flows

Our focus is on benign activities, but the dataset also includes attack traffic. These attacks are concentrated in the first two weeks of the dataset. Therefore, we decide not to use the first two weeks of traffic and concentrate only on the data from weeks 3 and 4. All flows originating from external sources outside the network are labeled as "suspicious" during this period. We discard these suspicious flows to maintain the focus on benign traffic and only consider network flows recorded within the OpenStack environment.

Splitting the Dataset We use the traffic from weeks 3 and 4 because they are the only two weeks in the dataset where no attacks were conducted on the network. Week 3 serves as the training set for the generative model, while week 4 is used as the simulation result for comparison.

The experiment aims at training the generative models on week 3 to produce a synthetic week that is as close as possible to the traffic of week 3. We then compare the synthetic data to the traffic from week 4 to evaluate its realism. Suppose one of our generative models produces traffic as close to week 3 as week 4. In that case, we can deduce that the generative model can replace the best possible simulation and is thus a viable alternative to simulation.

Since some of our evaluation methods require a separate evaluation set (see Subsection 5.5.3.2), we also collect benign traffic from week 2 to serve as the evaluation set. The splitting procedure is illustrated in Figure 5.3

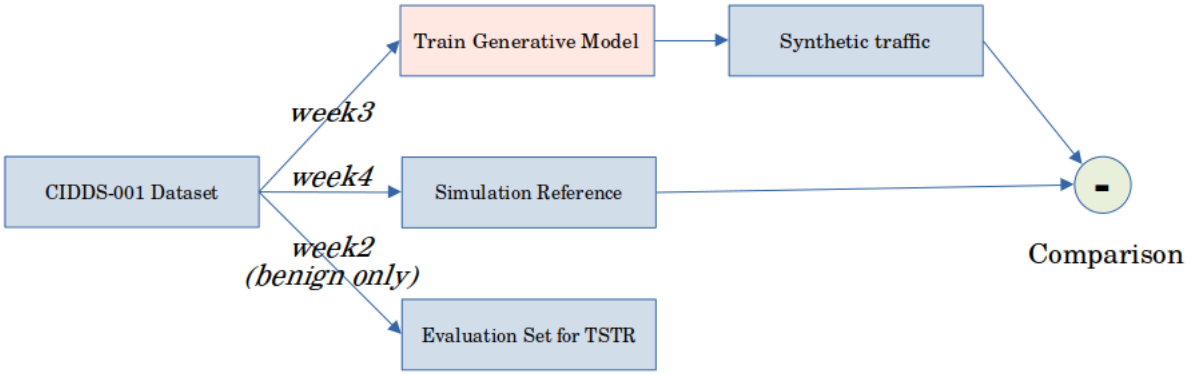

Figure 5.3: Splitting of the CIDDS datatet

5.5.2 Competing Methods

We decide to cover most of the options used in the state of the art for network flow generation. These models serve as baselines to compare against *FlowChronicle* and help address the first two research questions outlined in Subsection 5.1.

5.5.2.1 BN-based Methods

In Section 4.3, we introduce Bayesian Networks (BN) as a potential option for network flow generation. We, therefore, compare them to FlowChronicle in this part. We design two Bayesian Networks: one that samples individual flows and another that samples batches of flows.

- **IndependentBN**: This Bayesian Network is similar to the one used in Section 4.3. We use the same discretization process as *FlowChronicle*, making it equivalent to the BN_{bins} method of Section 4.5.
- **SequenceBN**: Instead of generating individual network flows, this model generates flows in batches of five. The preprocessing is similar to *FlowChronicle* and IndependentBN, but we generate 55 features (5 * 11) to produce five network flows simultaneously. This approach is meant to model the dependencies between the features of these five flows.

5.5.2.2 GAN-based Methods

We reuse the GAN presented in Subsection 4.5.3: CTGAN, E-WGAN-GP, and NetShare for our comparison with *FlowChronicle*. For more details on these models, please refer to Subsection 4.5.3. It is worth noting that among these models, NetShare is the only GAN that explicitly models temporal dependencies.

5.5.2.3 VAE-based Methods

We also include TVAE as a VAE-based method. TVAE is part of the same library as CTGAN [100] and is integrated similarly into our code. Like CTGAN, the model is trained for 300 epochs with the default hyperparameters.

5.5.2.4 Autoregressive Models / Transformers

Although we want to integrate STAN, we encounter difficulties with the implementation provided by the authors. Instead, we implement a generative pre-trained transformer (GPT) to generate sequences of discrete values representing various flows. We train the GPT model for 15 epochs, stopping when we observe the training loss converging.

We use the same discretization process outlined in Subsection 5.4.1 and trained GPT-2 to generate a new token based on the previous 60 tokens. After the training, the model could generate new flow features given the previous ones, allowing it to generate synthetic datasets sequentially.

5.5.3 Evaluation of Synthetic Traffic Quality

To support our claims, we need to evaluate the quality of synthetic flows individually and the preservation of temporal correlations present in the reference dataset in the synthetic dataset.

5.5.3.1 Independent Flows

To evaluate independent flows, we use the evaluation framework presented in Section 4.4 with a few modifications to DKC and MD.

DKC Because DKC is dataset-dependent, we need to adapt DKC to the current dataset. The various tests we execute to evaluate compliance are presented in Table 5.2.

Rule	Feature(s) concerned	Explanation
Protocol specific applications	Dst Pt; Proto	Some applications in the dataset use only one transport protocol
No outgoing data on ping request	Proto; Out Byte; Out Packet	The dataset recording was configured so that only reply were captured
No incoming data on NetBios or SSDP	Proto; In Byte; In Packet	Dataset specific
NetBios or SSDP only toward broadcast	Dst Pt; Dst IP Addr	Same reason as above
No incoming web traffic on local network	Dst Pt; Dst IP Addr	Outside web user reach the inner networks only through the web server
No mail traffic coming from outside	Dst Pt; Dst IP Addr	The only mail traffic recorded is the one initiated by internal user
All DNS requests are directed toward DNS server	Dst Pt; Dst IP Addr	A DNS server has been set up and it receives all the internal requests
TCP Flags only on TCP traffic	Flags; Proto	From Ring et al. [16]
The packet should have a minimal and a maximal size	In Byte; Out Byte; In Packet; Out Packet	From Ring et al. [16]
No negative Duration	Duration	The Duration of a Flow should always be positive
Only flows of 1 packet should have a null Duration	In Packet, Out Packet, Duration	The Duration is the summation of inter arrival times

Table 5.2: DKC compliance test

MD In Section 4.4, we introduce MD to evaluate the novelty of the generated data. Higher MD values indicate that more data is potentially copied from the training data. However, since many network flows in real-world datasets are copies of other flows, we aim for generative models to replicate the novelty seen in a reference dataset.

In this experiment, the reference dataset used for evaluating novelty is unavailable, so we revert to the initial idea of Goncalves et al. [83] of minimizing MD for the most novel generation. This implies that simulation will likely have the worst novelty score, as the generation process for both the training dataset and the simulation set is the same.

5.5.3.2 Temporal Dependency Preservation

Our previous evaluation framework focused on assessing independent network flows and did not account for preserving temporal dependencies. To demonstrate that *FlowChronicle* can preserve temporal dependencies present in the reference dataset, we separate categorical features from numerical features and treat each feature as a time series.

We consider the network flow dataset as a multidimensional time series, similar to the approach used in NetShare. In this framework, a network flow at time step i is represented as:

$$\mathbf{X}_i = \left(X_i^1, X_i^2, \ldots, X_i^n\right)$$

where $\mathbf{X}_i$ consists of n features (e.g., source IP, destination IP, packet count, etc.) observed at the i-th time step. The entire network flow dataset over T time steps is defined as:

$$\mathcal{D} = (\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T)$$

where $\mathcal{D}$ represents T network flows, each described by n features. Let $\mathbf{X}^j$ denote the time series of the j-th feature, where:

$$\mathbf{X}^j = \left(X_1^j, X_2^j, \ldots, X_T^j\right)$$

represents the values of the j-th feature over T time steps.

To assess the preservation of temporal dependencies, we examine the temporal dependencies of each feature $\mathbf{X}^j$ independently. This process involves identifying temporal dependencies in the real dataset and verifying that they are preserved in the generated dataset. Ideally, we would also evaluate interactions between different features over time to capture a more comprehensive temporal structure. However, due to computational expense and limited resources, this full analysis was not feasible (see Subsection 5.7.3).

Numerical Features For numerical features, inspired by Liao et al. [110] and Wiese et al. [111], we evaluate temporal dependencies between two time series by comparing their Autocorrelation Functions (ACF) [112]. The ACF of a feature $\mathbf{X}^j$ measures the linear correlation between X_t^j and X_{t+l}^j, where l is the lag. To do this, we compute the difference between the ACFs of the real and generated time series across all lags. However, as explained in Subsection 3.2.1, comparing ACF over all lags may mask essential differences at significant lags, as not all lags exhibit strong autocorrelation. To avoid this, we focus only on lags where the autocorrelation exceeds

the Bartlett confidence interval [113], ensuring that we evaluate temporal dependencies only where meaningful autocorrelations exist in the real dataset. This process is applied across all numerical features. Slight differences in ACFs suggest that the time series exhibit similar temporal structures, while more significant discrepancies indicate that temporal dependencies are not fully captured.

Categorical Features The autocorrelation method does not apply to categorical features. Instead, we could compare the contingency matrices of $P(X^j_{\text{real},t})|X^j_{\text{real},t-l})$ and $P(X^j_{\text{generated},t})|X^j_{\text{generated},t-l})$ across all lags l. However, computing these differences over millions of lags is computationally prohibitive. Inspired by Yoon et al.'s [114] predictive score, we implement the TSTR (Train on Synthetic, Test on Real) method [101] to assess the preservation of temporal dependencies in categorical features.

In this approach, we compare the performance of an LSTM[5] trained on real data versus one trained on synthetic data for a feature-wise autoregressive task, where the model predicts the current value of a feature based on its previous values. For each categorical feature, we first encode its values as a one-hot vector and train an LSTM to predict the next value in the sequence based on a context window of prior values. The first model is trained on the real training dataset (*week3*), while the second LSTM, with identical hyperparameters, is trained on the generated data. We then evaluate the accuracy of both models on benign traffic from *week2* (see Subsection 5.5.1.3). The TSTR score is the difference in accuracy between the two models.

This process is repeated for every categorical feature and is illustrated in Figure 5.4. We repeat the experiment multiple times with varying hyperparameters to prevent the score from depending too heavily on a single LSTM configuration. The final score for each categorical feature is the average accuracy difference across all LSTM configurations.

5.6 Results of the Experiment

In this section, we present the results of the experiment designed to evaluate the performance of *FlowChronicle* in comparison to other synthetic traffic generation methods and simulation. We start by assessing the quality of individual flows in Subsection 5.6.1, followed by an analysis of temporal dependency preservation in Subsection 5.6.2. Next, we explore the ability of *FlowChronicle* to generate adaptable traffic in Subsection 5.6.3, and finally, we compare the com-

[5] A Long Short-Term Memory (LSTM) network is a type of recurrent neural network capable of learning long-term dependencies, often used in tasks involving sequential data.

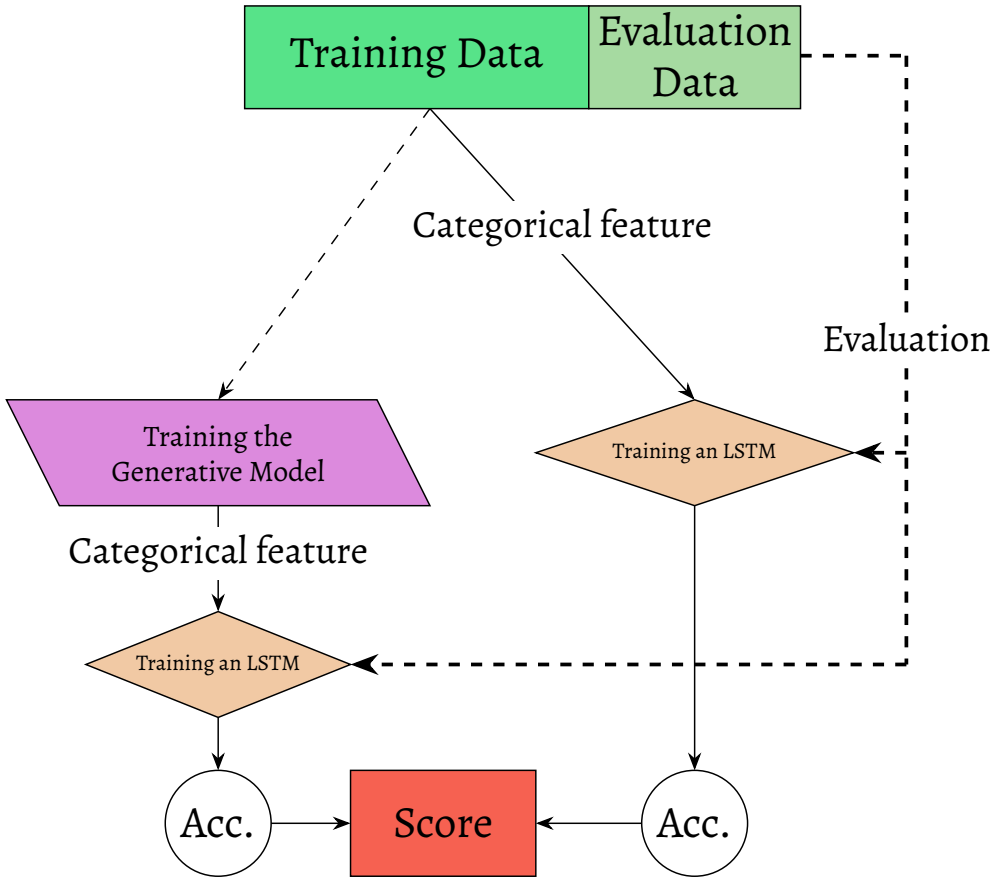

Figure 5.4: TSTR data pipeline

putational cost of *FlowChronicle* to other methods in Subsection 5.6.4. We conclude by summarizing our responses to the research questions in Subsection 5.6.5.

5.6.1 Independent Flows

We begin by evaluating the flows independently, without considering temporal dependencies between flows. The metrics were computed 20 times on 20 different subsamples of both the generated and training data. Each subsample includes 10,000 flows (like in Section 4.6). The median of each metric and the ranking of all models according to each metric are reported in Table 5.3. We present the result according to the four criteria of our benchmark : realism, diversity, novelty, compliance.

	Density	CMD	PCD	EMD	JSD	Coverage	DKC	MD	Rank
	Real. ↑	*Real.* ↓	*Real.* ↓	*Real./Div.* ↓	*Real./Div.* ↓	*Div.* ↑	*Comp.* ↓	*Nov.* ↓	*Average Ranking*
Simulation	**1 (0.69)**	2 (0.05)	2 (1.42)	**1 (0.00)**	2 (0.15)	**1 (0.59)**	**1 (0.00)**	9 (6.73)	**2.375**
IndependentBN	8 (0.24)	6 (0.23)	7 (2.73)	6 (0.11)	4 (0.27)	4 (0.38)	3 (0.05)	5 (5.47)	5.375
SequenceBN	7 (0.30)	3 (0.13)	6 (2.20)	5 (0.08)	3 (0.21)	3 (0.44)	2 (0.02)	6 (5.50)	4.375
TVAE	4 (0.50)	5 (0.18)	4 (1.77)	2 (0.01)	5 (0.30)	5 (0.33)	4 (0.07)	4 (5.18)	4.125
CTGAN	3 (0.56)	4 (0.15)	3 (1.56)	2 (0.01)	2 (0.15)	2 (0.51)	5 (0.11)	7 (5.69)	3.5
E-WGAN-GP	9 (0.02)	8 (0.34)	9 (3.70)	2 (0.01)	7 (0.38)	8 (0.02)	4 (0.07)	3 (4.66)	6.25
NetShare	6 (0.32)	7 (0.28)	**1 (1.36)**	4 (0.03)	6 (0.36)	6 (0.22)	3 (0.05)	**1 (3.73)**	4.25
Transformer	2 (0.65)	9 (0.78)	8 (3.65)	**1 (0.00)**	8 (0.55)	7 (0.03)	3 (0.05)	2 (3.76)	5
FlowChronicle	5 (0.41)	**1 (0.03)**	5 (2.01)	3 (0.02)	**1 (0.10)**	**1 (0.59)**	2 (0.02)	8 (5.87)	3.25

Table 5.3: Ranking of our different models without considering the preservation of temporal dependencies. For each metric, the median of the score is given between parentheses. Real.: Realism, Div.: Diversity, Comp.: Compliance, Nov.: Novelty, ↓: Lower is better, ↑: Higher is better. the best model for each metric is indicated in **bold**.

5.6.1.1 Realism

The Transformer and CTGAN models achieve relatively high Density scores (0.65 and 0.56, respectively). However, the Transformer model struggles to represent cross-feature correlations, as indicated by its CMD and PCD scores (0.78 and 3.65, respectively). Additionally, E-WGAN-GP generates data with low realism (Density: 0.02, CMD: 0.34, and PCD: 3.70), which could be due to the bidirectional nature of the dataset, whereas IP2Vec was initially designed for unidirectional datasets. *FlowChronicle* generates above-average results in terms of realism for synthetic network flows. As expected, the best-performing method in terms of realism is Simulation.

5.6.1.2 Diversity

While the Transformer produces realistic results, it struggles with diversity, showing a low Coverage score (0.03) and a high JSD (0.55). This can be attributed to a known behavior in autoregressive generative models called *degeneration* [115], [116], where the model learns to generate a specific sequence of network flows and repeats it during the generation process (see Table 5.4).

Date first seen	Proto	Src IP Addr	Dst IP Addr	Dst Pt	In Byte	In Packet	Out Byte	Out Packet	Flags
2017-04-05 00:05:00.262	UDP	192.168.200.9	DNS	53.0	0	0	1065	1	.....
2017-04-05 00:05:00.263	UDP	192.168.200.9	DNS	53.0	0	0	1002	1	.....
2017-04-05 00:05:01.264	UDP	192.168.200.9	DNS	53.0	0	0	1033	1	.....
2017-04-05 00:05:15.495	UDP	192.168.200.9	DNS	53.0	0	0	1057	1	.....
2017-04-05 00:05:17.802	UDP	192.168.200.9	DNS	53.0	0	0	1083	1	.....
2017-04-05 00:05:25.036	UDP	192.168.200.9	DNS	53.0	0	0	1016	1	.....
2017-04-05 00:05:25.038	UDP	192.168.200.9	DNS	53.0	0	0	1056	1	.....
2017-04-05 00:05:25.046	UDP	192.168.200.9	DNS	53.0	0	0	1068	1	.....

Table 5.4: Example of Generated Network Flows. Here, we observe that the Transformer baseline replicates a single network flow across an entire sequence. This tendency to repeatedly generate the same valid data point is known as *degeneration*.

We also notice that Bayesian Networks struggle to handle numerical variables, as reflected in the EMD scores (0.08 and 0.011 for SequenceBN and IndependentBN, respectively), a phenomenon already highlighted in Section 4.6. *FlowChronicle* and CTGAN are among the best models for covering the entire training distribution, and *FlowChronicle* performs almost on par with Simulation in this regard.

5.6.1.3 Compliance

Besides CTGAN (DKC: 0.11), most models can generate traffic that complies with the predefined rules. *FlowChronicle*, SequenceBN, and Simulation produce data with the fewest compliance issues.

5.6.1.4 Novelty

Both Simulation and *FlowChronicle* exhibit high MD scores (6.73 and 5.87, respectively), highlighting the fact that generating realistic network flows often requires copying parts of the training data, as discussed in Subsection 5.5.3.1. Methods that generated unrealistic data, such as Transformer (3.76) and NetShare (3.73), have lower MD scores, underscoring the realism/novelty tradeoff.

5.6.1.5 Overall

This evaluation shows that Simulation produces the best overall traffic, which was expected because the same simulation process was used to generate *week3* and *week4*. *FlowChronicle* emerges as the generative method closest to Simulation, with CTGAN following closely behind. Thus, we can affirmatively answer the first research question: *Does FlowChronicle produce better quality independent flows than other solutions?*

5.6.2 Preservation of Temporal Correlation

5.6.2.1 Categorical Features: Impact of Generated Sequences on the Accuracy of an LSTM

In Figure 5.5, we show the difference in accuracy between two similar LSTMs trained on the training data and synthetic data generated by each model for every categorical feature in the dataset. We observe that the two methods with the lowest difference across all categorical features are Simulation and *FlowChronicle*, indicating that *FlowChronicle* successfully learned the temporal dependencies for these features.

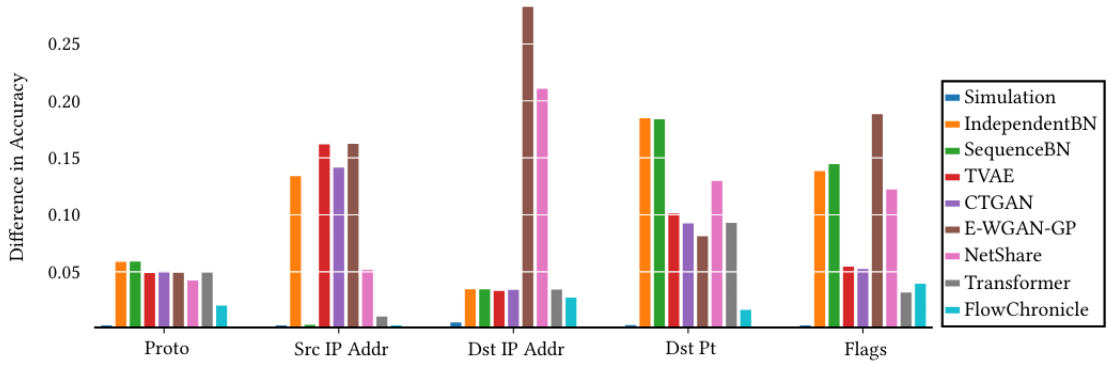

Figure 5.5: Average difference in accuracy of various LSTMs trained on the training data and synthetic data generated by different models. Each subgroup represents one feature, and each bar represents one generation method. Lower is better.

5.6.2.2 Numerical Features: Difference in Autocorrelation Functions (ACF)

In Figure 5.6, we present the differences in ACF across all numerical features between the real training dataset and the generated dataset. CTGAN and TVAE perform the worst in preserving temporal dependencies in numerical features, likely because both models come from the same library [100] and do not explicitly model temporal dependencies.

On the other hand, Transformer and NetShare preserve temporal dependencies pretty well since these models are designed to handle such dependencies. Surprisingly, E-WGAN-GP, which samples network flows independently, also shows relatively low differences. *FlowChronicle* outperforms these models, reproducing the autocorrelations across numerical features well. The difference in ACFs between the training set and the dataset generated by *FlowChronicle* is nearly equivalent to the difference between the training set and the Simulation set for the five numerical features. This demonstrates that *FlowChronicle* effectively learned the temporal dependencies in the training set for these features.

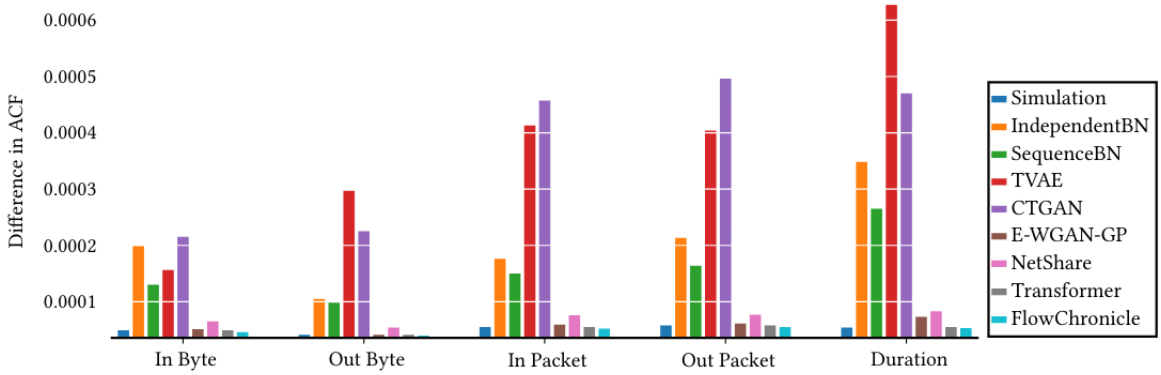

Figure 5.6: Difference in ACF between the generated data and the training data across all numerical features for various generative methods. Lower is better.

5.6.2.3 Overall

We can conclude that *FlowChronicle* is better than the other generative models for preserving temporal dependencies, both for categorical and numerical features. It is the only one that achieves a similarity level to that of simulation across all features. Thus, we can positively answer the second research question: *Does FlowChronicle better preserve temporal dependencies?*

5.6.3 Explainable Multi-Flow Patterns Discovered by *FlowChronicle* for an Adaptable Generation

To address the fourth research question outlined in Subsection 5.1, we explore whether *FlowChronicle* can generate specific network activities, adapting its output to meet each user's unique requirements.

The model of *FlowChronicle* is composed of patterns with associated frequencies, allowing users to adjust these frequencies to prioritize particular network activities in the generated

dataset, without retraining. This feature enables customization, as users can enrich the generation with patterns that suit their specific needs. This adaptability is not achievable with current simulation solutions [9], [12]. In simulations, users must rerun the entire process to increase the frequency of specific traffic types. With *FlowChronicle*, users can simply select and amplify the patterns associated with the desired activities, generating them as needed.

However, for this capability to be effective, *FlowChronicle* must learn patterns that can clearly be linked to identifiable network activities. With minimal expert knowledge, users can select patterns that correspond to their desired network activities, provided these patterns are interpretable. We identify several such patterns and present examples here. Since our focus is on sequential patterns (i.e., patterns involving multiple flows), we present only multi-flow patterns here. If an activity is associated with a single flow (e.g., a PING request), it can be modeled directly using a Bayesian Network, as discussed in Section 4.3.

The first pattern contains three partial flows, all from the same source IP to the same destination IP. All three partial flows are HTTPS requests (TCP Protocol, Destination Port 443). This pattern is likely generated when a browser requests a webpage that requires multiple resources (e.g., images, scripts, styles) from the same server, each fetched through a separate HTTPS connection.

A second pattern represents a DNS request (UDP protocol, destination port 53) followed by an HTTP flow (TCP protocol, destination port 80). The source IP remains the same for both flows. This is a classic network pattern: before a device can send an HTTP request to access a domain, it must resolve the domain's IP address. While the IP address could be cached locally, a DNS request may still be required to obtain it.

The third pattern includes two partial flows from the same source IP to different destination IPs. The first partial flow is an HTTPS request (TCP protocol, destination port 443), followed by a DNS request (UDP protocol, destination port 53). This can be explained by the fact that a webpage might include resources hosted on external servers (e.g., scripts or images). In such cases, the browser must send a DNS request to resolve the IP address of the external server. This pattern may be missing subsequent HTTPS connections following the DNS request. Interestingly, we do not find any multi-flow patterns related to non-Web protocols.

These patterns provide strong evidence that *FlowChronicle* is capable of learning explainable patterns that align with typical network behavior, which experts can validate. This allows us to constrain the generation to certain activities. We add more related patterns into our generated data if we want more of certain activities. This allows for an adaptable generation that is unreachable by simulation and can be a strong argument for favoring synthetic data over

simulated data for creating a network dataset (see Subsection 1.2.3 for more details).

However, to achieve truly adaptable generation, it is necessary to map more of our patterns to specific network activities. Although we observe patterns associated with certain network activities, most of patterns identified by the model lack clear explanations. Moreover, we did not examine the types of activities in the CIDDS-001 dataset, leaving the potential for adaptable generation present but currently unexplored.

5.6.4 Computational Cost

Comparing computational costs allows us to address the final objective of Subsection 5.1, which concerns comparing the efficiency of *FlowChronicle* to the one of simulation. In Table 5.5, we report the time taken for training each method and generating synthetic data from it. All experiments were conducted on a server with 500 GB of RAM, 2 AMD EPYC 7413 CPUs, and 3 A40 Nvidia GPUs.

One drawback of *FlowChronicle* is the time required for training and generating data. While *FlowChronicle* achieved the best performance on independent and temporal metrics, it also had the most prolonged training and generation times. Although long training times are a known issue of MDL-based methods, we are confident that the generation time could be significantly reduced due to the simplicity of the process. Reducing the time required to train the pattern miner and generate new data is a potential area of improvement for future work (see Subsection 5.7.3).

Simulation, however, is by far the most time-consuming process for a generation. Simulations typically require 168 hours to run for a week of traffic generation. While there is no explicit training time involved, since simulation methods are not trained models, an equivalent time-consuming step would be the creation of a detailed simulation setup. As a result, our comparison with simulation focuses solely on the generation time, as the preparation and creation of the simulation environment can vary significantly and is not directly comparable to *FlowChronicle*'s training phase.

However, generation speed alone does not fully address the question of efficiency, which, as defined in Section 4.1, refers to minimizing computational power while achieving high-quality output. Therefore, to assess efficiency, we also need to compare the cost of generation between Simulation and *FlowChronicle*. To achieve this, we examine the cost of renting a cloud infrastructure to run the generation process, estimating the expense of producing a week of traffic with both *FlowChronicle* and Simulation.

To generate a week of traffic with Simulation, one needs to configure the OpenStack en-

	Duration (hh:mm)	
Model	*Training*	*Generating*
Simulation	-	168:00
IndependentBN	00:12	$< 0{:}01$
SequenceBN	00:31	$< 0{:}01$
CTGAN*	29:12	00:02
TVAE*	02:01	00:03
E-WGAN-GP*	00:36	01:59
NetShare*	59:39	05:00
Transformer*	84:02	34:41
FlowChronicle	106:54	85:16

Table 5.5: Training and generating runtimes for producing one week of traffic. Methods annotated with * rely on GPU.

vironment and run it for 168 hours as detailed in the original dataset paper [6]. This requires setting up one external server, four internal servers, three Windows clients, three printer emulators, and 16 Linux clients. Based on AWS EC2 pricing, each terminal can be emulated on a VM using a small instance like t3.medium[6], which costs approximately $0.04 per hour at the time of writing. The cost of simulating the entire infrastructure for one week and recording the traffic would be approximately:

$$\$0.04 \times (1 + 4 + 3 + 3 + 16) \times 168 = \$181.44$$

Running the generation process with *FlowChronicle* takes less time but requires more powerful instances. Based on our experiments, an r6g.16xlarge instance (same number of CPU cores and RAM)[7] could reproduce the generation duration. The on-demand cost of this instance is approximately $3.22 per hour at the time of writing [8]. Thus, the cost of generating one week of traffic with *FlowChronicle* would be approximately:

$$86 \times \$3.22 = \$276.92$$

From this, we can see that *FlowChronicle* is more expensive than Simulation, even though it is faster to generate. It is important to note that the cost of generating traffic in simulations depends on the number of hosts in the network, which is not directly the case for *FlowChronicle*. As discussed in Section 1.2, this limitation impacts the scalability of simulations. If the

[6] https://aws.amazon.com/ec2/instance-types/t3/
[7] https://aws.amazon.com/fr/ec2/instance-types/r6g/
[8] https://instances.vantage.sh/aws/ec2/r6g.16xlarge

training dataset had contained more hosts, the cost of generating a week of traffic using simulation would have increased. In contrast, the generation cost for *FlowChronicle* is not directly influenced by the number of hosts.

5.6.5 Response to the research questions

Below we sum the answers to questions raised in Subsection 5.1 and respond to them individually

- **Does *FlowChronicle* produce better-quality independent flows than existing methods?** Our method is the best among the generative methods presented and closely approaches the quality of the Simulation for individual flows.
- **Does *FlowChronicle* better preserve temporal dependencies?** We have also seen that our method produces the most realistic dataset for preserving temporal dependencies among the generative methods presented, similar to Simulation.
- **Is the traffic generated by *FlowChronicle* better than simulated traffic?** The response to that question is less straightforward. We have shown that *FlowChronicle* can produce network flows that are close to Simulation in terms of quality, while Simulation was the best traffic overall. However, our simulation is a best-case scenario, where the emulating process perfectly mimics the activities recorded in the training dataset. In a more usual case, it is highly probable that the simulation activities would not match the real activities, lowering the realism of the resulting traffic. In such a configuration, *FlowChronicle* could be a better alternative than simulation for representing the real dataset. This will need further investigation to be shown.
- **Does *FlowChronicle* allow for an adaptable generation ?** We have seen that *FlowChronicle* is able to have an adaptable generation and that certain network activity can be generated without having to retrain the model or regenerate the entire traffic. However, this aspect needs to be further investigated in order to have a proper mapping of more of our patterns in terms of network activities
- **Is *FlowChronicle* more efficient?** While *FlowChronicle* is faster than Simulation in generating network flows, it incurs higher computational costs, which makes it less efficient overall. This underscores the need for further optimization to improve cost efficiency. Reducing the cost to generate remains the main area for enhancing *FlowChronicle*, but

improving the speed of its training process—which is longer than that of other generative methods—could also be beneficial.

5.7 Limitations and Future Work

In this section, we address the limitations of *FlowChronicle* and the experimental setup used to evaluate it, followed by potential directions for future improvements.

5.7.1 Limitations of *FlowChronicle*

The first limitation of *FlowChronicle* is the expressiveness of the pattern language. While capable of capturing simple patterns, the current language struggles with complex behaviors, such as repeating flows in video streaming. A more expressive language could represent these behaviors, but this would significantly increase the computational complexity and search space.

Additionally, the greedy search algorithm, although effective, is time-consuming due to the vast number of possible pattern combinations. Potential improvements, such as parallelization, could reduce runtime by splitting the dataset into independent chunks for processing. However, ensuring consistency across chunks and managing cross-chunk dependencies presents challenges that remain unresolved.

5.7.2 Experimental Limitations

The experimental setup also has limitations. In comparing *FlowChronicle* with a simulation, we used an idealized simulation scenario that assumed perfect alignment between the simulated and training datasets. In reality, simulations often oversimplify user interactions, which may lead to less realistic traffic. A more comprehensive evaluation would involve comparing *FlowChronicle* against a wider range of real and simulated datasets to account for this variability. Additionally, the CIDDS-001 dataset lacks labels linking flows to network activities, limiting our ability to verify if detected patterns correspond accurately to specific activities.

Finally, while we compared the computational cost of *FlowChronicle* and simulation, the analysis was based on a simplified cost model using on-demand pricing. A more fine-grained cost comparison, incorporating factors such as reservation-based pricing and hardware optimizations, could better reflect the true cost differences between the two approaches.

5.7.3 Future Work

Building on the limitations discussed, several areas of future work can further enhance *FlowChronicle*'s capabilities in synthetic network traffic generation. Below are key directions for future improvements.

5.7.3.1 Expanding the Pattern Language

While the current pattern language is effective at detecting simple patterns, it may fall short in modeling more complex behaviors, such as repeating flows in video streaming or long-duration activities. Future research should focus on developing a more expressive pattern language that can model repeating sequences or longer-term dependencies common in real-world network activities.

A challenge will be balancing the expressiveness of the language with the complexity of the search space. Incorporating a dataset labeled with specific network activities, such as the CIC IoT dataset 2023[9], would allow further validation of an enhanced version of *FlowChronicle*. Successfully identifying and reproducing behaviors linked to user activity would demonstrate the pattern mining approach's ability to capture complex behaviors, further proving *FlowChronicle*'s adaptability.

5.7.3.2 Improving Temporal Dependency Evaluation

Current evaluations of temporal dependencies in *FlowChronicle* focus primarily on single-feature dependencies. Future work should focus on developing new metrics that assess multi-feature temporal correlations to provide a more comprehensive assessment of synthetic traffic's temporal dynamics.

One potential direction is to explore the use of Multivariate Dependent Dynamic Time Warping (MVDTW) [117], which aims to measure the alignment of temporal sequences across multiple features. Although MVDTW is still a relatively new metric in time series evaluation, further research could establish its consistency and applicability for evaluating complex network traffic patterns.

[9] https://www.unb.ca/cic/datasets/iotdataset-2023.html

5.7.3.3 Parallelization and Efficiency

The time-consuming nature of *FlowChronicle*'s greedy search for patterns can be addressed through parallelization techniques. Dividing the workload across multiple processors can speed up both the training and pattern discovery phases, potentially overcoming the computational bottlenecks seen in larger datasets or more complex patterns.

Additionally, differential search techniques, which dynamically adjust the search space based on prior results, could help reduce redundant computations. Another promising direction would be investigating lightweight or approximate search methods that could significantly lower the computational cost of pattern discovery without compromising the quality of generated data. These methods would make *FlowChronicle* more practical for real-time or large-scale traffic generation scenarios.

5.7.3.4 Exploring Large Language Models (LLMs) for Synthetic Data Generation

Given the success of Large Language Models (LLMs) in generating consistent data sequences across various domains, LLMs present a promising alternative for synthetic network traffic generation. Their ability to capture long-range dependencies and maintain internal consistencies throughout data sequences makes them particularly suitable for generating network flows with complex temporal structures.

Future work could investigate how LLMs can enhance the realism of synthetic traffic generated by *FlowChronicle*. Additionally, LLMs like GPT could offer improvements in terms of training and generation efficiency, potentially reducing the time and resources needed to generate high-quality synthetic data.

5.7.3.5 Real-World Validation

While *FlowChronicle* has demonstrated strong performance in controlled experiments, it still requires validation in real-world environments. Future research should focus on applying *FlowChronicle* to real-world datasets that reflect the unpredictability and diversity of network traffic encountered in practical deployments.

Moreover, conducting experiments in environments where traditional simulation methods fail to capture real-world traffic complexity would provide further insights into *FlowChronicle*'s performance. Validating *FlowChronicle* in these challenging scenarios could demonstrate its potential superiority over conventional simulation methods, especially in environments with highly dynamic and variable traffic patterns.

Chapter 6

Conclusion

To evaluate NIDS, access to benign network traffic is essential, as it provides the baseline for distinguishing between normal and malicious activity, yet sharing real network traffic raises significant privacy concerns. Real network traffic frequently contains sensitive user data. While anonymization is a potential solution, it is difficult to implement and poses some challenges, like the risk of de-anonymization for example. Moreover, accurately labeling real traffic requires expert knowledge, and errors in this process can negatively impact NIDS performance, increasing false positives or false negatives.

Simulations have been proposed as an alternative to real traffic, but they present their own challenges. Modifying simulations to accommodate new types of traffic or changes in initial conditions requires rerunning the entire simulation, a time-consuming and resource-intensive process. Furthermore, simulating large networks increases resource demands, as it involves modeling many agents and processes, which drives up computational costs and complexity.

Synthetic data generation is often promoted as a solution that addresses the limitations of both real traffic and simulation. The generation of "fake" traffic is considered privacy-compliant and easy to label. Moreover, synthetic traffic generation is seen as scalable, as it theoretically does not depend on network size, and adaptable, as it allows new traffic to be generated without requiring the entire process to be rerun. These characteristics suggest that synthetic traffic generation could be a viable alternative to both real traffic and simulation.

The primary goal of this thesis is to develop a synthetic network traffic generation system capable of replacing simulation for NIDS evaluation. The system aims to generate scalable and adaptable traffic while preserving critical characteristics of real network traffic, such as temporal dependencies, diversity and compliance. In this chapter, we first introduce our contributions in this area and then discuss potential future work. We address current limitations in our approach and explore possible extensions to broaden the scope of our research.

6.1 Contributions of this thesis

6.1.1 Review of Synthetic Traffic Generation Methods

The first major contribution of this thesis is a comprehensive review of existing synthetic traffic generation methods, offering a detailed overview of the current state of the art while identifying areas that require improvement. One of the key findings from this review is the limitations of Generative Adversarial Networks (GAN) in the context of network flow generation.

GAN, while widely used for various generative tasks, exhibits several shortcomings when applied to network flow generation. GAN often struggle to capture the complex dependencies between network flow features, which is crucial for accurately representing realistic traffic. Moreover, training GAN is time-consuming.

6.1.2 Bayesian Networks as an Alternative

A key contribution of this thesis is the proposal of Bayesian Networks (BN) as an alternative to GAN for synthetic network flow generation. BN offer several advantages, particularly in their ability to accurately capture inter-feature dependencies in tabular network flow data. BN are inherently designed to model these relationships, making them particularly well-suited for generating realistic network flows.

Our experiments showed that BN outperform GAN on smaller datasets and achieve comparable performance on larger datasets while requiring significantly fewer computational resources. This demonstrates that traditional machine learning methods, such as BN, can compete with, and in some cases surpass, more complex neural network architectures when synthesizing network flow data. We argue that BN should be considered more extensively in future research on synthetic network flow generation.

6.1.3 Development of a Unified Benchmark for Evaluation

The third contribution of this thesis is the introduction of a comprehensive benchmark for evaluating synthetic traffic generation methods.

Current generative methods lack a unified framework for evaluating the quality of the generated traffic across multiple dimensions. Typically, evaluations rely on the utility of the traffic for NIDS, but this approach fails to provide a holistic view of the quality of the traffic itself. For example, utility metrics do not account for phenomena such as mode collapse, where a generative model fails to produce the full diversity of traffic patterns observed in real data. Likewise,

they do not address training data duplication, which can lead to privacy violations if sensitive information is unintentionally replicated from the training dataset. Mode invention, where models generate unrealistic or artificial patterns not present in the real data, also goes unnoticed.

To address these limitations, we introduce a comprehensive evaluation framework based on key criteria necessary for synthetic traffic generation. The benchmark assesses synthetic network flow generation across several critical dimensions, including:

- **Realism**: The generated network flows should closely resemble the training data, appearing as if they were sampled from the same underlying distribution.
- **Diversity**: The generated flows should exhibit the same level of variability as the training ones, avoiding overfitting or repetitive patterns.
- **Novelty**: While being realistic, the synthetic flows should not simply replicate the training data. They must introduce sufficient variations to ensure the traffic is new and not a direct copy of the original dataset.
- **Compliance**: The generated flows should adhere to protocol specifications, ensuring that all generated traffic is valid and follows established network protocols.

This benchmark enables researchers to assess and compare synthetic traffic generation methods without making assumptions about the intended use of the generated traffic. It provides an objective means to evaluate the quality of traffic across these critical criteria. Additionally, the benchmark helps identify and address issues such as mode collapse, data copying, and mode invention. By structuring the evaluation process around these dimensions, the benchmark allows for fair and systematic comparisons between different generative approaches, ensuring a thorough assessment of their strengths and limitations. We also extend this same benchmark to include an evaluation of how well our generated dataset captures temporal dependencies present in the real dataset.

6.1.4 Preserving Temporal Dependencies in Synthetic Network Flows

Another significant contribution of this thesis is addressing the challenge of preserving temporal dependencies in synthetic network traffic. In real-world scenarios, user activities often

result in sequential patterns, such as a DNS request followed by a visit to a website. These temporal dependencies are crucial for generating realistic traffic, as they reflect natural user behavior and interaction flows. However, many current generative methods produce network flows independently, thereby missing these essential patterns, which diminishes the realism of the generated traffic.

To resolve this issue, we developed *FlowChronicle*, a novel approach that combines pattern mining with Bayesian Networks to capture both independent and sequential flow patterns. *FlowChronicle* outperforms existing methods in two critical areas: it generates high-quality individual flows while also preserving temporal dependencies between them. Our experiments demonstrate that *FlowChronicle* offers a substantial improvement over alternative generative models and has the potential to be more adaptable than traditional simulation methods. We believe that *FlowChronicle* is one of the first advanced network flow generation methods capable of generating high-quality flows while maintaining realistic sequential patterns, making it a valuable tool for both research and practical applications in network traffic generation.

6.2 Future Work

This section proposes several directions to advance synthetic network traffic generation. These directions are divided into *future work to address current limitations* and *future work to broaden the research scope*, each offering specific avenues for overcoming existing constraints and exploring novel applications.

6.2.1 Addressing Current Limitations

This section discusses future work aimed at addressing the limitations identified in *FlowChronicle*, including improvements in model performances, scalability, and practical applicability.

6.2.1.1 Optimizing Cardinality and Discretization in Model Comparison

Our comparison between GAN and BN, discussed in Section 4.5, reveals limitations due to restricted cardinality and the discretization of numerical data. For discrete variables, the limited cardinality used in our experiments was chosen to maintain model efficiency but may have constrained GAN' performance by simplifying the variable representation. Future studies should explore the effect of increasing cardinality, particularly for discrete features such as port numbers, which could enhance GAN' ability to model complex network behaviors. This exploration

would provide a more accurate comparison between GAN and BN and clarify their respective computational trade-offs and performance scalability.

The discretization approach for numerical data in Bayesian Networks was similarly guided by efficiency considerations to reduce model complexity. However, since discretization significantly impacts the accuracy of Bayesian Networks, future work should investigate optimal discretization strategies that better capture continuous variables in network data. Future studies could provide a more balanced comparison by systematically assessing the effects of different discretization approaches on BN.

6.2.1.2 Evaluating Dataset Size on Model Performance

The scalability of BN has been assessed based on feature count, yet the relationship between dataset size (i.e., number of flows) and the performance of BN remains largely unexplored. Although larger datasets could potentially enable neural network-based models, such as GAN, to better capture inter-feature dependencies, the datasets we used are of a common size widely employed in GAN studies. This ensures that our comparisons are meaningful, even if future work might explore whether larger datasets further enhance GAN performance.

Similarly, *FlowChronicle* would benefit from a study on how training dataset length affects model quality and training time, especially when employing parallelization strategies. In this setup, smaller datasets are processed in parallel and then recombined. Initial experiments show that parallelization can effectively reduce computational demands: increasing the number of chunks from 50 to 100 reduced training time from 18 hours 30 minutes to 8 hours 5 minutes without compromising temporal correlations. However, splitting data into too many chunks (e.g., 400) led to an over-representation of single-flow patterns, indicating that too few flows per chunk can disrupt temporal dependencies. Therefore, determining the optimal chunk size to balance computational cost with temporal fidelity remains unresolved.

Two key questions arise from these observations:

- Could GAN outperform BN on larger datasets by capturing feature dependencies more effectively in abundant data?
- What is the optimal chunk size that balances generated dataset quality and training time of *FlowChronicle* in a parallelized setup?

Addressing these questions would provide valuable insights into the scalability of generative methods, a crucial factor given the scarcity of real benign network data.

6.2.1.3 Extending Beyond Network Flows to Packet Captures

Currently, *FlowChronicle* generates network flows exclusively. While network flows capture essential information, they impose a fixed set of flow-level features on the user. Expanding synthetic generation to include packet captures (pcaps) would allow users greater flexibility by enabling them to craft their own feature sets from packet-level data, which provides for finer-grained details such as packet sizes, timestamps, and sequences.

As discussed in Subsection 3.1.3, the primary challenges in generating realistic synthetic pcaps include creating valid packet payloads and accurately modeling header sequences within a flow. The first challenge could be approached by using automata to structure packet sequences or by enhancing the network flow feature set to incorporate header-level details. Recent research, such as [89] and [17], provides a basis for modeling realistic header attributes. Generating packet payloads should ideally occur as a subsequent step. Since many packet payloads are encrypted, future work could explore simulating encrypted payloads with padding. This approach would maintain the packet structure while safeguarding data privacy. These advancements would make *FlowChronicle* more adaptable, allowing users to define features according to specific research or application needs.

6.2.2 Broadening the Research Scope

This section describes directions for expanding *FlowChronicle* beyond current limitations, exploring novel methodologies, and introducing new applications in synthetic data generation.

6.2.2.1 Exploring Large Language Models (LLMs) for Network Flow Generation

Large Language Models (LLMs) have demonstrated the capacity to generalize across various tasks, including previously unseen applications, as highlighted in [118]. Advanced LLMs are pre-trained on extensive datasets, unlike the GPT-2 model we use in Section 5.5. This pretraining enables them to generalize and adapt across various domains, including the novel network flow data generation task.

In preliminary studies that we conducted, LLMs (LLaMA-2-70B, Mistral-7B, Llama-2-7B, and ChatGPT4) are provided with network flow datasets in CSV format and prompted with: "Above is X minutes of network traffic in CSV format. Each line corresponds to one network flow. Can you generate X more minutes of traffic using the same format?" The variable X is adjusted to fit within the model's maximum context window. These initial experiments reveal several challenges, which are outlined below.

Challenges Encountered One major issue is that the LLMs often fail to recognize when to stop generating data, leading to unbounded output. Additionally, smaller models, such as Mistral-7B and Llama-2-7B, encounter difficulties with context reinitialization, where they lose track of their objective when the output exceeds their token limit. Another significant limitation involves the models' restricted context sizes, which constrain the quantity of traffic that could be processed or generated within a single pass.

Proposed Solutions To address these issues, prompt engineering was employed, including inserting flags at the start and end of the input data to delineate the generation boundaries clearly. This approach helped guide the models to terminate the generation more appropriately. For example, a prompt like: "Above is X minutes of network traffic in CSV format. Each line corresponds to one network flow. Can you generate X more minutes of traffic using the same format?" was used. Additionally, the context reinitialization issue was mitigated by ending each prompt with a specific termination flag, prompting the model to complete the task and add a final "End of generated traffic" marker, improving task consistency. Another example of a prompt with flags given to LLMs is provided in Figure 6.1:

```
 ## Start of Original Traffic
Date first seen, Proto, Src IP Addr, Dst IP Addr, Dst Pt, In Byte, Out Byte
2017-04-05 00:00:00.266, TCP, 192.168.220.15, 10031_250, 445.0, 743.0, 547.0
2017-04-05 00:00:01.264, UDP, 192.168.200.4, 192.168.200.255, 137.0, 0, 0.0
2017-04-05 00:00:10.534, TCP, 192.168.220.4, 192.168.100.5, 445.0, 97.0, 522.0
2017-04-05 00:00:11.275, UDP, 192.168.200.5, 192.168.200.255, 138.0, 0, 0.0
2017-04-05 00:00:25.558, TCP, 10031_250, 192.168.100.5, 445.0, 248.0, 128.0
2017-04-05 00:01:00.096, UDP, 192.168.200.8, 10031_250, 1900.0, 0, 0.0
...
2017-04-05 00:05:57.461, TCP, 10031_250, 192.168.100.5, 445.0, 787.0, 138.0
2017-04-05 00:06:03.715, UDP, 192.168.200.9, 192.168.200.255, 138.0, 0, 0.0
## End of Original Traffic
```

Above is network traffic in a .csv format. Each line corresponds to one network flow. Can you generate the same kind of traffic using the same format as the original traffic but with different timestamps and traffic details? The generated traffic should be in the same time range and similar to the original one.

```
 ## Start of Generated Traffic
```

Figure 6.1: Prompt used to generate network flows with LLMs. In this example, the flows are reduced for readability, showing only a portion of the original traffic. The traffic is encapsulated by start and termination flags.

To overcome the context size limitations, we propose two potential approaches. The first is to develop a custom encoding that captures network flow structures more efficiently and train a foundational model specifically for network flow data. This would allow us to define specialized tokens representing network flows more compactly, similar to the transformer baseline in

Section 5.5. However, this would require significant computational resources and further research. The second, more immediate approach involves adopting a Retrieval-Augmented Generation (RAG) strategy. The model could retrieve relevant chunks as needed by splitting the dataset into manageable chunks stored in a vector database, effectively expanding its context size through memory retrieval.

6.2.2.2 Privacy Considerations in Network Traffic Generation

Although this thesis evaluates Novelty to prevent direct data replication, more advanced privacy metrics are needed to protect against sensitive data leakage. Specifically, more advanced membership inference attacks, as discussed in [119], could test whether a synthetic dataset unintentionally reveals information about the original data. Additionally, implementing differential privacy techniques, such as those explored in [120], could add an extra layer of protection by ensuring that individual data points are not identifiable. By incorporating these techniques, future work could align *FlowChronicle* with privacy standards essential for security-sensitive applications.

6.2.2.3 Testing Generated Traffic on NIDS

Applying synthetic traffic generated by *FlowChronicle* to real NIDS, such as ntopng[1] or others [121], [122], would validate its practical applicability. This process involves generating benign traffic that minimizes false positives on flow-based NIDS. By evaluating *FlowChronicle*'s performance across different types of NIDS, future work could demonstrate its versatility and applicability for real-world intrusion detection scenarios. Testing on multiple flow-based NIDS with varying feature requirements would also highlight *FlowChronicle*'s flexibility to diverse detection systems.

6.2.2.4 Enhancing the Pattern Language in *FlowChronicle*

The current pattern language used in *FlowChronicle* faces challenges when representing complex, recurring flow patterns, such as those in video streaming or repeated communication sequences in IoT traffic. Expanding the pattern language to include more expressive, nuanced flow sequences could better capture these types of traffic patterns, increasing the synthetic data's realism and utility in NIDS testing and other applications.

[1] https://www.ntop.org/guides/ntopng/index.html

To verify this, an activity identification experiment should be conducted to evaluate whether *FlowChronicle* can detect complex network activities. This would require a dataset with labels linking specific network flows to particular activities. Initial research suggests that the CIC IoT dataset 2023[2] may be well-suited for this purpose.

This research direction would likely require advanced pattern-matching techniques, possibly incorporating non-greedy search algorithms to optimize pattern extraction without substantially increasing computational overhead. By balancing the model's descriptive power with computational efficiency, a more expressive pattern language could enhance *FlowChronicle* 's applicability across diverse synthetic traffic scenarios.

6.2.2.5 Creating and Sharing a Comprehensive Synthetic Dataset for NIDS Evaluation

A long-term goal is to establish a large-scale, comprehensive synthetic NIDS evaluation dataset comprising both benign and malicious traffic. Such a dataset would serve as a benchmark, offering a synthetic alternative to traditional simulated datasets. Including both benign and realistic malicious traffic patterns would allow the dataset to bridge the gap between synthetic data generation and traditional simulations, potentially positioning *FlowChronicle* as a foundational tool for security-focused research and practical NIDS deployment.

Generating synthetic malicious traffic presents unique challenges, particularly in merging attack traffic with benign traffic while preserving the properties of both. Successfully combining these traffic types requires careful handling to ensure that malicious characteristics accurately reflect real-world attack profiles without distorting the integrity of benign flows [23]. Future work should explore methods for integrating diverse attack vectors, such as DDoS, malware signatures, and network reconnaissance behaviors, into synthetic traffic.

While current evaluation benchmarks focus primarily on benign flows, future research should establish benchmarks and evaluation metrics tailored to datasets containing both benign and malicious traffic, as discussed in [123]. By developing and sharing such datasets, *FlowChronicle* could substantially contribute to synthetic traffic generation in network security, enhancing its applicability across various security-focused research and practical implementations.

[2] https://www.unb.ca/cic/datasets/iotdataset-2023.html

Bibliography

[1] S. Northcutt and J. Novak, *Network Intrusion Detection: An Analyst's Handbook*, 3rd. USA: New Riders Publishing, 2002, ISBN: 0735712654 (page 15).

[2] M. Roesch, "Snort - lightweight intrusion detection for networks", in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99, Seattle, Washington: USENIX Association, 1999, pp. 229–238 (page 16).

[3] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection", *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 3, pp. 186–205, Aug. 2000, ISSN: 1094-9224. DOI: 10.1145/357830.357849. [Online]. Available: https://doi.org/10.1145/357830.357849 (page 16).

[4] P. Voigt and A. Bussche, *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Jan. 2017, ISBN: 978-3-319-57958-0. DOI: 10.1007/978-3-319-57959-7 (page 17).

[5] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set", in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528 (page 17).

[6] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection", 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3637071 (pages 17, 18, 74, 138).

[7] A. H. Lashkari, G. Draper Gil, M. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features", in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP)*, SciTePress, 2017, pp. 253–262 (pages 17, 27–29).

[8] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", in *International Conference on Information Systems Security and Privacy*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4707749 (pages 17, 74, 82, 84).

[9] S. Abt and H. Baier, "A plea for utilising synthetic data when performing machine learning based cybersecurity experiments", in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, ser. AISec '14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 37–45, ISBN: 9781450331531. DOI: 10.1145/2666652.2666663. [Online]. Available: https://doi.org/10.1145/2666652.2666663 (pages 17, 18, 39, 136).

[10] T. J. Anande and M. S. Leeson, "Generative adversarial networks (gans): a survey on network traffic generation", *International Journal of Machine Learning and Computing*, vol. 12, pp. 333–343, Oct. 2022. DOI: 10.18178/ijmlc.2022.12.6.1120. [Online]. Available: https://api.semanticscholar.org/CorpusID:253339791 (pages 18, 46).

[11] H. Navidan, P. Fard Moshiri, M. Nabati, *et al.*, "Generative adversarial networks (gans) in networking: a comprehensive survey evaluation", *Computer Networks*, vol. 194, p. 108 149, May 2021. DOI: 10.1016/j.comnet.2021.108149 (pages 18, 29).

[12] C. Pandey, V. Tiwari, R. S. Rathore, R. H. Jhaveri, D. S. Roy, and S. Selvarajan, "Resource-efficient synthetic data generation for performance evaluation in mobile edge computing over 5g networks", *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1866–1878, 2023. DOI: 10.1109/OJCOMS.2023.3306039 (pages 18, 136).

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks", *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020 (pages 19, 32).

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners", 2019 (page 19).

[15] R. Singh, H. Kumar, and R. K. Singla, "A reference dataset for network traffic activity based intrusion detection system", *Int. J. Comput. Commun. Control*, vol. 10, pp. 390–402, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:53976879 (page 19).

[16] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks", *Computers Security*, vol. 82, pp. 156–172, 2019 (pages 19, 42, 45, 66, 67, 81, 82, 87, 89, 128).

[17] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative deep learning for internet of things network traffic generation", in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, 2020, pp. 70–79 (pages 19, 148).

[18] X. Jiang, S. Liu, A. Gember-Jacobson, *et al.*, "Netdiffusion: network data augmentation through protocol-constrained traffic generation", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 8, no. 1, pp. 1–32, 2024 (pages 19, 53).

[19] S. Xu, M. Marwah, M. Arlitt, and N. Ramakrishnan, "Stan: synthetic network traffic generation with generative neural models", in Sep. 2021, pp. 3–29, ISBN: 978-3-030-87838-2. DOI: 10.1007/978-3-030-87839-9_1 (pages 19, 29, 47, 58, 64–68, 78, 89).

[20] R. Sommer and V. Paxson, "Outside the closed world: on using machine learning for network intrusion detection", in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316. DOI: 10.1109/SP.2010.25 (page 21).

[21] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan", in *Data Mining and Big Data*, Y. Tan and Y. Shi, Eds., Singapore: Springer Nature Singapore, 2022, pp. 409–423, ISBN: 978-981-19-8991-9 (page 21).

[22] Q. Li, J. Li, Y. Li, F. Jiu, and Y. Chu, "An adaptive enhancement method of malicious traffic samples based on dcgan-resnet system", *International Journal of Information Technologies and Systems Approach*, vol. 17, pp. 1–17, Jan. 2024. DOI: 10.4018/IJITSA.343317 (page 21).

[23] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser, "Id2t: a diy dataset creation toolkit for intrusion detection systems", in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015, pp. 739–740. DOI: 10.1109/CNS.2015.7346912 (pages 21, 151).

[24] H. Zimmermann, "Osi reference model - the iso model of architecture for open systems interconnection", *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980. DOI: 10.1109/TCOM.1980.1094702 (page 24).

[25] J. Postel, *Internet protocol*, RFC 791, 1981. [Online]. Available: https://tools.ietf.org/html/rfc791 (page 25).

[26] S. Deering and R. Hinden, *Internet protocol, version 6 (ipv6) specification*, 2017. [Online]. Available: https://datatracker.ietf.org/doc/rfc8200/ (page 25).

[27] R. Hinden and S. Deering, *Ipv6 addressing architecture*, RFC 4291, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4291 (page 25).

[28] M. Larsen and F. Gont, *Recommendations for Transport-Protocol Port Randomization*, RFC 6056, Jan. 2011. DOI: 10.17487/RFC6056. [Online]. Available: https://www.rfc-editor.org/info/rfc6056 (pages 25, 125).

[29] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Pearson, 2011 (pages 25–27).

[30] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters", in *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, IEEE, 2003, pp. 44–51 (page 27).

[31] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 5th ed. Morgan Kaufmann, 2011 (page 27).

[32] W. Haag, "Nfdump tools for netflow data analysis", *https://github.com/phaag/nfdump*, 2014, Accessed: 2024-05-28. [Online]. Available: https://github.com/phaag/nfdump (page 27).

[33] L. Deri, "Nprobe: an open source netflow probe for gigabit networks", *Proceedings of the IEEE International Symposium on Computers and Communications (ISCC)*, pp. 200–205, 2003 (page 27).

[34] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning", *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 56–76, 2008 (page 28).

[35] B. Claise, *Cisco systems netflow services export version 9*, RFC 3954, 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954 (page 28).

[36] E. D. Kolaczyk and G. Csárdi, *Statistical Analysis of Network Data with R*. Springer, 2014 (page 28).

[37] R. Hofstede, P. Čeleda, B. Trammell, *et al.*, "Flow monitoring explained: from packet capture to data analysis with netflow and ipfix", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014. DOI: 10.1109/COMST.2014.2321898 (pages 28, 125).

[38] B. Claise, "Ip flow information export (ipfix) protocol specifications", *RFC 7011, Internet Engineering Task Force (IETF)*, 2013 (pages 28, 29).

[39] D. Olivier, M. Peterson, and S. Wright, *Cisco NetFlow: Advanced Network Monitoring for Performance and Security*. Cisco Press, 2004 (pages 28, 29).

[40] P. Phaal, S. Panchen, and N. McKee, "Sflow: real-time network monitoring for high-speed networks", *IEEE Network*, vol. 14, no. 5, pp. 35–42, 2001 (page 28).

[41] J. Networks, *Juniper networks j-flow monitoring and configuration guide*, https://www.juniper.net/documentation/en_US/junos/topics/concept/services-interfaces-jflow-overview.html, 2020 (pages 28, 29).

[42] H. Technologies, *Huawei netstream technology overview*, https://support.huawei.com/enterprise/en/doc/EDOC1100075466, 2018 (pages 28, 29).

[43] M. Marwah and M. Arlitt, "Deep learning for network traffic data", in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22, Washington DC, USA: Association for Computing Machinery, 2022, pp. 4804–4805, ISBN: 9781450393850. DOI: 10.1145/3534678.3542618. [Online]. Available: https://doi.org/10.1145/3534678.3542618 (page 29).

[44] G. Agrawal, A. Kaur, and S. Myneni, "A review of generative models in generating synthetic attack data for cybersecurity", *Electronics*, vol. 13, no. 2, 2024, ISSN: 2079-9292. [Online]. Available: https://www.mdpi.com/2079-9292/13/2/322 (page 29).

[45] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets", *Computers Security*, vol. 86, pp. 147–167, 2019, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.06.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481930118X (pages 29, 82, 84).

[46] S. Bourou, A. E. Saer, T.-H. Velivassaki, A. Voulkidis, and T. Zahariadis, "A review of tabular data synthesis using gans on an ids dataset", *Information*, vol. 12, no. 09, p. 375, 2021 (pages 29, 45, 46, 65, 67, 89).

[47] Y. Yin, Z. Lin, M. Jin, G. Fanti, and V. Sekar, "Practical gan-based synthetic ip header trace generation using netshare", in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 458–472 (pages 29, 47, 48, 53, 58, 64–68, 78, 87, 90).

[48] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016 (page 30).

[49] M. I. Jordan and T. M. Mitchell, "Machine learning: trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255–260, 2015 (page 30).

[50] G. Carleo, I. Cirac, K. Cranmer, *et al.*, "Machine learning and the physical sciences", *Reviews of Modern Physics*, vol. 91, no. 4, p. 045 002, 2019 (page 30).

[51] N. Radakovich, M. Nagy, and A. Nazha, "Machine learning in haematological malignancies", *The Lancet Haematology*, vol. 7, no. 7, e541–e550, 2020 (page 30).

[52] L. Wendlinger, E. Berndl, and M. Granitzer, "Methods for automatic machine-learning workflow analysis", in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part V 21*, Springer, 2021, pp. 52–67 (page 30).

[53] T. Guo, J. Xu, X. Yan, *et al.*, "Ease the process of machine learning with dataflow", in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 2437–2440 (page 30).

[54] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach", *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2017 (page 30).

[55] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, pp. 436–444, 2015 (page 31).

[56] G. Huang, S. Song, J. N. Gupta, and C. Wu, "Semi-supervised and unsupervised extreme learning machines", *IEEE transactions on cybernetics*, vol. 44, no. 12, pp. 2405–2417, 2014 (page 31).

[57] H. Chen, "Challenges and corresponding solutions of generative adversarial networks (gans): a survey study", *Journal of Physics: Conference Series*, vol. 1827, no. 1, p. 012 066, Mar. 2021. DOI: 10.1088/1742-6596/1827/1/012066. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1827/1/012066 (pages 32, 37).

[58] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: an overview", *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018 (page 33).

[59] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders", *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019 (page 33).

[60] M. Ehsan Abbasnejad, A. Dick, and A. van den Hengel, "Infinite variational autoencoder for semi-supervised learning", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5888–5897 (page 34).

[61] S. Zhao, J. Song, and S. Ermon, "Infovae: information maximizing variational autoencoders", *arXiv preprint arXiv:1706.02262*, 2017 (page 34).

[62] D. Heckerman, "Bayesian networks for data mining", *Data mining and knowledge discovery*, vol. 1, pp. 79–119, 1997 (pages 34, 74).

[63] G. Schwarz, "Estimating the Dimension of a Model", *Annals of Statistics*, vol. 6, no. 2, pp. 461–464, Jul. 1978 (page 34).

[64] N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham, "A survey of bayesian network structure learning", *Artificial Intelligence Review*, vol. 56, no. 8, pp. 8721–8814, 2023 (page 36).

[65] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: a survey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–21, Dec. 2022. DOI: 10.1109/TNNLS.2022.3229161 (pages 36, 37).

[66] H.-P. Kriegel, E. Schubert, and A. Zimek, "The (black) art of runtime evaluation: are we comparing algorithms or implementations?", *Knowledge and Information Systems*, vol. 52, pp. 341–378, 2017 (page 37).

[67] S. Huang, J. Li, J. Ye, *et al.*, "A sparse structure learning algorithm for gaussian bayesian network identification from high-dimensional data", *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 6, pp. 1328–1342, 2012 (page 37).

[68] A. Giuseppe, F. Giampaolo, C. Guida, *et al.*, "Synthetic and privacy-preserving traffic trace generation using generative ai models for training network intrusion detection systems", *Available at SSRN 4643250*, 2023 (page 39).

[69] X. Huang, X. Wang, Y. Liu, and Q. Xue, "A distributed traffic replay framework for network emulation", *Information*, vol. 14, no. 2, 2023, ISSN: 2078-2489. DOI: 10.3390/info14020059. [Online]. Available: https://www.mdpi.com/2078-2489/14/2/59 (page 39).

[70] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of machine learning algorithms for anomaly detection", in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1–8. DOI: 10.1109/CyberSecurity49315.2020.9138871 (page 39).

[71] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially learned anomaly detection", in *2018 IEEE International conference on data mining (ICDM)*, IEEE, 2018, pp. 727–736 (pages 40, 45, 61, 64).

[72] J. Donahue and K. Simonyan, "Large scale adversarial representation learning", *Advances in neural information processing systems*, vol. 32, 2019 (page 40).

[73] W. Xu, J. Jang-Jaccard, T. Liu, F. Sabrina, and J. Kwak, "Improved bidirectional gan-based approach for network intrusion detection using one-class classifier", *Computers*, vol. 11, no. 6, p. 85, 2022 (pages 41, 61, 64, 65, 67).

[74] T. Zixu, K. S. K. Liyanage, and M. Gurusamy, "Generative adversarial network and auto encoder based anomaly detection in distributed iot networks", in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, IEEE, 2020, pp. 1–7 (page 41).

[75] L. Han, Y. Sheng, and X. Zeng, "A packet-length-adjustable attention model based on bytes embedding using flow-wgan for smart cybersecurity", *IEEE Access*, vol. 7, pp. 82 913–82 926, 2019 (pages 41, 125).

[76] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks", in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Aug. 2017, pp. 214–223. [Online]. Available: https://proceedings.mlr.press/v70/arjovsky17a.html (pages 41, 45).

[77] V. Chetlapalli, H. Agrawal, K. Iyer, M. A. Gregory, V. Potdar, and R. Nejabati, "Performance evaluation of iot networks: a product density approach", *Computer Communications*, vol. 186, pp. 65–79, 2022, ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2022.01.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422000160 (page 42).

[78] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "Ip2vec: learning similarities between ip addresses", in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 657–666. DOI: 10.1109/ICDMW.2017.93 (pages 42–44).

[79] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium", in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6629–6640, ISBN: 9781510860964 (page 43).

[80] L. D. Manocchio, S. Layeghy, and M. Portmann, "Flowgan-synthetic network flow generation using generative adversarial networks", in *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*, IEEE, 2021, pp. 168–176 (pages 45, 46, 65, 67).

[81] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks", *Proc. VLDB Endow.*, vol. 11, no. 10, pp. 1071–1083, Jun. 2018, ISSN: 2150-8097. DOI: 10.14778/3231751.3231757. [Online]. Available: https://doi.org/10.14778/3231751.3231757 (pages 45, 89).

[82] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan", in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (pages 45, 87, 89).

[83] A. Gonçalves, P. Ray, B. Soper, J. Stevens, L. Coyle, and A. Sales, "Generation and evaluation of synthetic patient data", *BMC Medical Research Methodology*, vol. 20, May 2020. DOI: 10.1186/s12874-020-00977-1 (pages 46, 48, 54, 56, 57, 59, 60, 62–64, 70, 81, 128).

[84] Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar, "Using gans for sharing networked time series data: challenges, initial promise, and open questions", in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 464–483 (pages 47, 53, 65–68, 78).

[85] A. Cheng, "Pac-gan: packet generation of network traffic using generative adversarial networks", in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 2019, pp. 0728–0734 (pages 49, 66, 67).

[86] A. Meddahi, H. Drira, and A. Meddahi, "Sip-gan: generative adversarial networks for sip traffic generation", in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, IEEE, 2021, pp. 1–6 (pages 49, 66, 67).

[87] S. K. Nukavarapu, M. Ayyat, and T. Nadeem, "Miragenet-towards a gan-based framework for synthetic network traffic generation", in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 3089–3095 (page 49).

[88] B. Dowoo, Y. Jung, and C. Choi, "Pcapgan: packet capture file generator by style-based generative adversarial networks", in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, IEEE, 2019, pp. 1149–1154 (page 51).

[89] F. Meslet-Millet, S. Mouysset, and E. Chaput, "Necstgen: an approach for realistic network traffic generation using deep learning", in *GLOBECOM 2022-2022 IEEE Global Communications Conference*, IEEE, 2022, pp. 3108–3113 (pages 51, 148).

[90] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests", in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '04/Performance '04, New York, NY, USA: Association for Computing Machinery, 2004, p. 392, ISBN: 1581138733. DOI: 10.1145/1005686.1005733. [Online]. Available: https://doi.org/10.1145/1005686.1005733 (page 51).

[91] C. Rolland, J. Ridoux, and B. Baynat, "Litgen, a lightweight traffic generator: application to p2p and mail wireless traffic", in *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, ser. PAM'07, Louvain-la-Neuve, Belgium: Springer-Verlag, 2007, pp. 52–62, ISBN: 9783540716167 (page 51).

[92] I. E. Livieris, N. Alimpertis, G. Domalis, and D. Tsakalidis, "An evaluation framework for synthetic data generation models", in *Artificial Intelligence Applications and Innovations*, I. Maglogiannis, L. Iliadis, J. Macintyre, M. Avlonitis, and A. Papaleonidas, Eds., Cham: Springer Nature Switzerland, 2024, pp. 320–335, ISBN: 978-3-031-63219-8 (pages 55, 58, 61, 64).

[93] M. Hernandez, G. Epelde, A. Alberdi, R. Cilla, and D. Rankin, "Synthetic tabular data evaluation in the health domain covering resemblance, utility, and privacy dimensions", *Methods of information in medicine*, vol. 62, Jan. 2023. DOI: 10.1055/s-0042-1760247 (pages 55, 57–59, 61, 62, 64, 65).

[94] F. K. Dankar, M. K. Ibrahim, and L. Ismail, "A multi-dimensional evaluation of synthetic data generators", *IEEE Access*, vol. 10, pp. 11 147–11 158, 2022. DOI: 10.1109/ACCESS.2022.3144765 (pages 56–60, 64).

[95] M. F. Naeem, S. J. Oh, Y. Uh, Y. Choi, and J. Yoo, "Reliable fidelity and diversity metrics for generative models", in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 7176–7185. [Online]. Available: https://proceedings.mlr.press/v119/naeem20a.html (pages 56, 57, 61, 64, 80).

[96] A. Borji, "Pros and cons of gan evaluation measures", *Computer vision and image understanding*, vol. 179, pp. 41–65, 2019 (pages 56–58, 64).

[97] A. Borji, "Pros and cons of gan evaluation measures: new developments", *Computer Vision and Image Understanding*, vol. 215, p. 103 329, 2022 (pages 56, 57, 61, 64).

[98] C. Meehan, K. Chaudhuri, and S. Dasgupta, "A non-parametric test to detect data-copying in generative models", in *International Conference on Artificial Intelligence and Statistics*, 2020 (pages 57, 62, 64).

[99] A. Alaa, B. Van Breugel, E. S. Saveliev, and M. van der Schaar, "How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models", in *International Conference on Machine Learning*, PMLR, 2022, pp. 290–306 (pages 57, 61, 63, 64).

[100] N. Patki, R. Wedge, and K. Veeramachaneni, "The synthetic data vault", in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 399–410. DOI: 10.1109/DSAA.2016.49 (pages 58, 64, 127, 134).

[101] P. Zingo and A. Novocin, "Introducing the tstr metric to improve network traffic gans", in Apr. 2021, pp. 643–650, ISBN: 978-3-030-73099-4. DOI: 10.1007/978-3-030-73100-7_46 (pages 61, 64, 65, 67, 130).

[102] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "Ugr'16: a new dataset for the evaluation of cyclostationarity-based network idss", *Computers Security*, vol. 73, pp. 411–424, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.11.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404817302353 (pages 74, 85).

[103] M. Sarhan, S. Layeghy, and M. Portmann, "Towards a standard feature set for network intrusion detection system datasets", *Mob. Netw. Appl.*, vol. 27, no. 1, pp. 357–370, Feb. 2022, ISSN: 1383-469X. DOI: 10.1007/s11036-021-01843-0. [Online]. Available: https://doi.org/10.1007/s11036-021-01843-0 (page 84).

[104] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, "Error prevalence in nids datasets: a case study on cic-ids-2017 and cse-cic-ids-2018", in *2022 IEEE Conference on Communications and Network Security (CNS)*, 2022, pp. 254–262. DOI: 10.1109/CNS56114.2022.9947235 (page 84).

[105] M. Lanvin, P.-F. Gimenez, Y. Han, F. Majorczyk, L. Mé, and É. Totel, "Errors in the cicids2017 dataset and the significant differences in detection performances it makes", in *Risks and Security of Internet and Systems*, S. Kallel, M. Jmaiel, M. Zulkernine, A. Hadj Kacem, F. Cuppens, and N. Cuppens, Eds., Cham: Springer Nature Switzerland, 2023, pp. 18–33, ISBN: 978-3-031-31108-6 (page 84).

[106] P. A. Osorio-Marulanda, G. Epelde, M. Hernandez, I. Isasa, N. M. Reyes, and A. B. Iraola, "Privacy mechanisms and evaluation metrics for synthetic data generation: a systematic review", *IEEE Access*, vol. 12, pp. 88 048–88 074, 2024. DOI: 10.1109/ACCESS.2024.3417608 (page 107).

[107] P. Grünwald, *The Minimum Description Length Principle*. MIT Press, 2007 (page 114).

[108] P. Li and M. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997 (page 114).

[109] J. Rissanen, "A universal prior for integers and estimation by minimum description length", vol. 11, no. 2, pp. 416–431, 1983 (page 114).

[110] H. Ni, L. Szpruch, M. Sabate-Vidales, B. Xiao, M. Wiese, and S. Liao, "Sig-wasserstein gans for time series generation", in *Proceedings of the Second ACM International Conference on AI in Finance*, 2021, pp. 1–8 (page 129).

[111] R. K. Magnus Wiese Robert Knobloch and P. Kretschmer, "Quant gans: deep generation of financial time series", *Quantitative Finance*, vol. 20, no. 9, pp. 1419–1440, 2020. DOI: 10.1080/14697688.2020.1730426. eprint: https://doi.org/10.1080/14697688.2020.1730426. [Online]. Available: https://doi.org/10.1080/14697688.2020.1730426 (page 129).

[112] M. H. Naveed, U. S. Hashmi, N. Tajved, N. Sultan, and A. Imran, "Assessing deep generative models on time series network data", *IEEE Access*, vol. 10, pp. 64 601–64 617, 2022. DOI: 10.1109/ACCESS.2022.3177906 (page 129).

[113] M. S. Bartlett, "On the theoretical specification and sampling properties of autocorrelated time-series", *Supplement to the Journal of the Royal Statistical Society*, vol. 8, no. 1, pp. 27–41, 1946, ISSN: 14666162. [Online]. Available: http://www.jstor.org/stable/2983611 (visited on 05/16/2024) (page 130).

[114] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks", in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf (page 130).

[115] Z. Zhang, C. Gao, C. Xu, R. Miao, Q. Yang, and J. Shao, "Revisiting representation degeneration problem in language modeling", in *Findings*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226283524 (page 133).

[116] J. Gao, D. He, X. Tan, T. Qin, L. Wang, and T. Liu, "Representation degeneration problem in training natural language generation models", in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=SkEYojRqtm (page 133).

[117] E. Brophy, Z. Wang, Q. She, and T. Ward, "Generative adversarial networks in time series: a systematic literature review", *ACM Comput. Surv.*, vol. 55, no. 10, Feb. 2023, ISSN: 0360-0300. DOI: 10.1145/3559540. [Online]. Available: https://doi.org/10.1145/3559540 (page 141).

[118] L. C. Melo, "Transformers are meta-reinforcement learners", in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, Jul. 2022, pp. 15 340–15 359. [Online]. Available: https://proceedings.mlr.press/v162/melo22a.html (page 148).

[119] P. Francis, C. Berneanu, and E. Gashi, *Syndiffix: more accurate synthetic structured data*, 2023. arXiv: 2311.09628 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2311.09628 (page 150).

[120] D. Sun, J. Q. Chen, C. Gong, T. Wang, and Z. Li, *Netdpsyn: synthesizing network traces under differential privacy*, 2024. arXiv: 2409.05249 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2409.05249 (page 150).

[121] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "E-graphsage: a graph neural network based intrusion detection system for iot", in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, Budapest, Hungary: IEEE Press, 2022, pp. 1–9. DOI: 10.1109/NOMS54207.2022.9789878. [Online]. Available: https://doi.org/10.1109/NOMS54207.2022.9789878 (page 150).

[122] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for netflow and ipfix", Oct. 2013, pp. 227–234, ISBN: 978-3-901882-53-1. DOI: 10.1109/CNSM.2013.6727841 (page 150).

[123] R. Flood, G. Engelen, D. Aspinall, and L. Desmet, "Bad design smells in benchmark nids datasets", in *2024 IEEE 9th European Symposium on Security and Privacy (EuroSP)*, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2024, pp. 658–675. DOI: 10.1109/EuroSP60621.2024.00042. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EuroSP60621.2024.00042 (page 151).

[124] M. E. Tschuchnig, C. Ferner, and S. Wegenkittl, "Sequential iot data augmentation using generative adversarial networks", in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 4212–4216.

[125] G. Andresini, A. Appice, L. De Rose, and D. Malerba, "Gan augmentation to deal with imbalance in imaging-based intrusion detection", *Future Generation Computer Systems*, vol. 123, pp. 108–127, 2021.

[126] X. Jiang, S. Liu, A. Gember-Jacobson, P. Schmitt, F. Bronzino, and N. Feamster, "Generative, high-fidelity network traces", in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023, pp. 131–138.

[127] S. Hui, H. Wang, T. Li, *et al.*, "Large-scale urban cellular traffic generation via knowledge-enhanced gans with multi-periodic patterns", in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 4195–4206.

[128] R. H. Randhawa, N. Aslam, M. Alauthman, and H. Rafiq, "Evasion generative adversarial network for low data regimes", *IEEE Transactions on Artificial Intelligence*, 2022.

[129] G. Dlamini and M. Fahim, "Dgm: a data generative model to improve minority class presence in anomaly detection domain", *Neural Computing and Applications*, vol. 33, pp. 13 635–13 646, 2021.

[130] Z. Lin, Y. Shi, and Z. Xue, "Idsgan: generative adversarial networks for attack generation against intrusion detection", in *Pacific-asia conference on knowledge discovery and data mining*, Springer, 2022, pp. 79–91.

[131] P. Wang, S. Li, F. Ye, Z. Wang, and M. Zhang, "Packetcgan: exploratory study of class imbalance for encrypted traffic classification using cgan", in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–7.

[132] M. Kim, "Ml/cgan: network attack analysis using cgan as meta-learning", *IEEE Communications Letters*, vol. 25, no. 2, pp. 499–502, 2020.

[133] S. Fathi-Kazerooni and R. Rojas-Cessa, "Gan tunnel: network traffic steganography by using gans to counter internet traffic classifiers", *Ieee Access*, vol. 8, pp. 125 345–125 359, 2020.

[134] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[135] A. Orebaugh, G. Ramirez, J. Beale, and J. Wright, *Wireshark Ethereal Network Protocol Analyzer Toolkit*. Syngress Publishing, 2007, ISBN: 1597490733.

[136] V. Jacobson, C. Leres, and S. McCanne, "Tcpdump: a protocol analyzer for the 4.3bsd unix operating system", in *Proc. of the Winter 1989 USENIX Conference*, USENIX Association, 1989, pp. 317–329.

[137] F. Ratle, G. Camps-Valls, and J. Weston, "Semisupervised neural networks for efficient hyperspectral image classification", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 5, pp. 2271–2282, 2010.

[138] G.-J. Qi and J. Luo, "Small data challenges in big data era: a survey of recent progress on unsupervised and semi-supervised methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 2168–2187, 2020.

[139] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges", *Cybersecurity*, vol. 2, 2019. [Online]. Available: %5Curl%7Bhttps://api.semanticscholar.org/CorpusID:197645932%7D.

[140] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning bayesian networks: the combination of knowledge and statistical data", *Machine learning*, vol. 20, pp. 197–243, 1995.

[141] S. Kundu, *Fundamentals of Computer Networks*. PHI Learning Pvt. Ltd., 2008.

[142] T. M. Chen, "Network traffic modeling", in *The handbook of computer networks*, vol. 3, Wiley Hoboken, NJ, 2007, p. 156.

[143] P. T. Watrobski and D. H. Summerville, "De-encapsulation of network packets for network protocol reverse engineering", in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, pp. 557–562. DOI: 10.1109/MILCOM.2016.7795386.

[144] M. Hassan, M. E. Haque, M. E. Tozal, V. Raghavan, and R. Agrawal, "Intrusion detection using payload embeddings", *IEEE Access*, vol. 10, pp. 4015–4030, 2021.

[145] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters", in *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, IEEE, 2003, pp. 44–51.

[146] D. A. Freedman, *Statistical Models: Theory and Practice*. Cambridge University Press, 2009.

[147] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. CRC Press, 1984.

[148] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley Sons, 2009.

[149] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.

[150] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures", 2006.

[151] S. Nakayama and D. Watling, "Consistent formulation of network equilibrium with stochastic flows", *Transportation Research Part B-methodological*, vol. 66, pp. 50–69, 2014. DOI: 10.1016/J.TRB.2014.03.007.

[152] S. Layeghy, M. Gallagher, and M. Portmann, "Benchmarking the benchmark — comparing synthetic and real-world network ids datasets", *Journal of Information Security and Applications*, vol. 80, p. 103 689, 2024, ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2023.103689. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212623002739.

[153] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests", *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, p. 392, Jun. 2004, ISSN: 0163-5999. DOI: 10.1145/1012888.1005733. [Online]. Available: https://doi.org/10.1145/1012888.1005733.

[154] R. Flood and D. Aspinall, "Measuring the complexity of benchmark nids datasets via spectral analysis", in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2024, pp. 335–341. DOI: 10.1109/EuroSPW61312.2024.00043. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EuroSPW61312.2024.00043.

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

CentraleSupélec

Titre : Génération par apprentissage automatique de trafic réseau pour l'évaluation des outils de détection d'intrusion

Mot clés : Réseaux, Génération de trafic, Détection d'intrusion, Intelligence Artificielle

Résumé : Avec la montée en puissance des infrastructures numériques, la cybersécurité est devenue une priorité mondiale. Les systèmes de détection d'intrusion réseau (NIDS) sont essentiels pour sécuriser les communications en détectant les activités malveillantes. Cependant, pour évaluer l'efficacité des NIDS, il est nécessaire de disposer de grands volumes de trafic réseau bénin. Or, l'acquisition de ces données pose des problèmes de confidentialité, d'étiquetage, et les méthodes de simulation de trafic atteignent leurs limites. Face à ces défis, la recherche s'oriente vers la génération de trafic synthétique, qui permet de protéger la vie privée et de passer à l'échelle, mais souffre encore d'un manque de cadre standard pour évaluer la qualité des données générées, et son efficacité par rapport aux simulations traditionnelles reste à démontrer.

Cette thèse s'inscrit dans cet effort de remplacement des simulations par des approches de génération de trafic synthétique. Nous commençons par un état de l'art détaillé des méthodes existantes, en mettant en lumière leurs tendances et leurs limites. Constatant l'absence d'un cadre d'évaluation standardisé, nous proposons un protocole d'évaluation permettant de mesurer la qualité du trafic généré par différents modèles. Par ailleurs, nous introduisons deux nouvelles approches de génération de flux réseau : la première, basée sur des réseaux bayésiens, se montre plus performante que les méthodes actuelles reposant sur des GAN ; la seconde, fondée sur la découverte de motifs récurrents, produit un trafic diversifié et réaliste, offrant ainsi une alternative prometteuse aux simulations traditionnelles.

Ces contributions visent à positionner la génération de trafic synthétique comme une solution crédible pour remplacer les simulations, en fournissant à la communauté de la sécurité des réseaux des outils plus efficaces et adaptés à la création de jeux de données synthétiques de qualité. Le protocole d'évaluation et les méthodes proposées représentent un pas important vers une évaluation plus rigoureuse et cohérente des techniques de génération de trafic.

Title: Network Traffic Generation for Evaluation of Intrusion Detection Tools

Keywords: Network, Synthetic Traffic Generation, Machine Learning, Intrusion Detection

Abstract: The increasing reliance on digital infrastructures has made cybersecurity a critical global concern. Network Intrusion Detection Systems (NIDS) play a vital role in safeguarding network communications by detecting malicious activities. However, evaluating the effectiveness of NIDS requires large, representative datasets of benign network traffic, which are difficult to obtain due to privacy concerns, labeling challenges, and the limitations of simulated traffic. To address these challenges, the research community

has turned to model-based synthetic data generation, which offers privacy preservation and scalability but lacks comprehensive evaluation standards and proven effectiveness over traditional simulation methods.

This thesis contributes to the ongoing effort to replace simulation with model-based network traffic generation. We first conduct a comprehensive survey of model-based methods, highlighting current trends and limitations. Recognizing the absence of a standardized evaluation framework, we develop a benchmark for assessing the quality of generated traffic across various generative models. Furthermore, we propose two novel methods for generating network flows: one based on Bayesian Networks that outperforms existing GAN-based methods, and another based on Pattern Mining that produces realistic, diverse network traffic. The latter method offers the potential to substitute traditional simulation in network traffic generation, particularly for NIDS evaluation.

Through these contributions, we aim to establish model-based generation as a viable alternative to simulation, providing the network security community with more efficient and scalable tools for creating high-quality synthetic datasets. Our proposed benchmark and generation methods represent a significant step towards this goal, facilitating more rigorous and meaningful comparisons in future research.