CascadeNet: Generating Network Traffic with High-Fidelity Temporal Patterns

Runwei Lu[1] Yanran Deng[1] Ruixuan Li[1] Jinting Liu[1] Yuejie Wang[2]
Xinyu Li[3] Deming Xu[1] Han Tian[4] Kai Chen[5] Guyue Liu[2]

[1]*New York University Shanghai* [2]*Peking University* [3]*Carnegie Mellon University*
[4]*University of Science and Technology of China* [5]*Hong Kong University of Science and Technology*

Abstract

Facing the challenge of limited network trace access, the exploration of synthetic trace generation has become crucial for research. Although current methods manage to replicate the statistical characteristics of network traffic accurately, they fail to capture the temporal dynamics of network activities. This gap stems primarily from their approach to data representation. To address this issue, we propose a novel representation of network traces by aggregating network flows into time series. Built upon this data representation, we propose CascadeNet, an end-to-end framework embedded with CascadeGAN—a hierarchical generative model—to generate network traffic with high-fidelity temporal patterns while learning complex flow structures and dependencies. We also develop several techniques to facilitate the transformation from aggregated time series to timestamps. Our evaluations across four diverse IPv4 header traces show (1) CascadeNet surpasses baselines by 41%~76% on temporal distance metrics; (2) CascadeNet outperforms baselines in downstream tasks; (3) it offers remarkable scalability, reducing training time by 7.3×~25× compared to state-of-the-art method.

1 Introduction

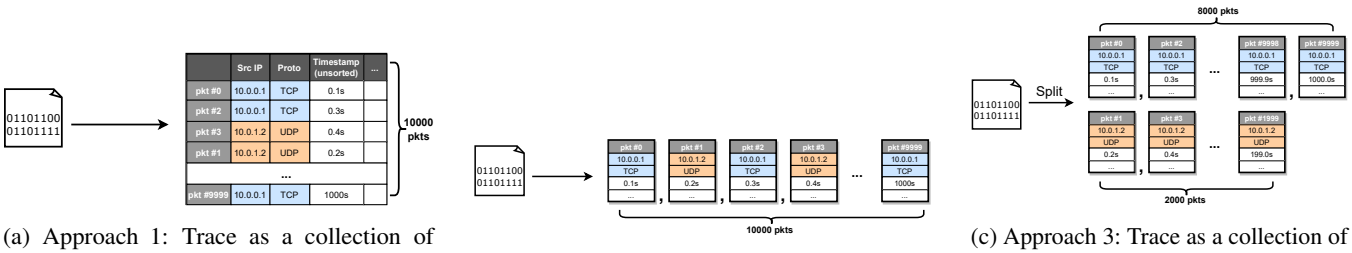
Data-driven methodologies in network research and development have gained significant momentum in recent years [17, 18]. Key to these researches is the availability of high-fidelity network traces that accurately reflect the diversity of real-world network traffic. However, access to such traces is often limited due to privacy concerns and proprietary restrictions. This challenge has led to an increased interest in synthetic network trace generation as a viable alternative for research and development purposes. Various methods have been proposed to generate synthetic network traces, including simulation-based methods [2, 26], model-driven methods [31, 37] and machine learning-driven methods [11, 16, 28, 41, 45, 47]. Machine learning models can automatically learn data patterns from real network traces and generate high-fidelity synthetic traces, making them less reliant on domain expertise and human effort compared to other methods. As a result, machine learning-based approaches have emerged as a practical and promising solution.

Although ML-based methods have shown great capability, we find it a common flaw that they struggle to reproduce how network activity and behavior evolve over time, which we refer to as *temporal patterns*. This causes the results of the synthetic traces to deviate from the real traces in many downstream tasks, compromising the fidelity requirement of synthetic traces generation. For example, in burst analysis, researchers need to examine the instantaneous changes in throughput within the traffic to understand the network dynamics. In machine learning-based anomaly detection, features such as the interarrival time between packets or the number of packets per second are also considered crucial to distinguish anomalies. Since low-fidelity synthetic traces can mislead researchers into drawing incorrect conclusions, it is a pressing issue that needs to be addressed.

The fundamental limitation arises from the *data representation* strategies adopted by existing approaches. For example, E-WGAN-GP [28] models traces as unordered collections of records, thereby disregarding inter-record dependencies and temporal correlations. STAN [45] represents traces as sequences of records, compelling the model to infer temporal dynamics from excessively long sequences, which is an inherently challenging task. NetShare [47] introduces the notion of flow separation, yet continues to treat each flow as a record sequence. Consequently, despite reduced sequence lengths, such representations remain inadequate to effectively capture temporal patterns.

In this work, we introduce CascadeNet, an end-to-end framework designed to generate network traces with high-fidelity temporal patterns. In designing CascadeNet, we have tackled the following key challenges:

- *Implicit Temporal Patterns in Network Traces*: We observe that a key challenge for machine learning models to learn temporal patterns in network traces is that they are typically implicit within the network traces, such as throughput



(a) Approach 1: Trace as a collection of records

(b) Approach 2: Trace as a sequence of records

(c) Approach 3: Trace as a collection of flows, flow as a sequence of records

Figure 1: Different data representation methods for network trace.

and packet rate. These patterns emerge from the aggregation of features across multiple records within the network trace. To overcome this challenge, we propose a novel data representation method that aggregates each flow into a fixed-length time series. This method explicitly presents temporal patterns to machine learning models, helping them generate synthetic traces with higher fidelity.

- *Heterogeneity among Different Data Components*: The novel data representation also introduces challenges in selecting machine learning models. Here, network traces are divided into three components: IP 5-tuples as identifiers for each flow, aggregated time series representing temporal patterns, and record fields. The heterogeneity of these components complicates the use of a single off-the-shelf model to generate them, while using three separate models would disrupt their interdependencies. To address this challenge, we developed CascadeGAN, a hierarchical generative model that effectively captures the features of the three components and their relationships. CascadeGAN consists of three generators that generate IP 5-tuples, aggregated time series, and record fields in a coarse-to-fine sequence, along with three discriminators to help the generators learn the patterns of real data.
- *Temporal Patterns Reflection in Synthetic Traces*: While the novel data representation method and hierarchical generative model are able to effectively capture the temporal patterns, we still need to generate the timestamp of each record to reflect these patterns. Therefore, we developed several timestamp generation techniques, from equidistant generation to ML-based generation, to reproduce the temporal patterns with the same or even higher granularity than the aggregated time series. This is the final step in ensuring that CascadeNet produces synthetic traces with high-fidelity temporal patterns.

Moreover, we observed that the skewed data distribution in network traces presents significant challenges for machine learning models. To address this, we applied various optimization techniques including the zero-inflation method and conditional input strategy to improve the models' capability, ensuring the synthetic traces accurately replicate the patterns within the real ones.

Overall, CascadeNet achieves 41%~76% better accuracy on temporal distribution metrics. In downstream tasks, CascadeNet can also maintain better result consistency with real data. Moreover, CascadeNet reduces the training and generating time for 7.3×~25× and 9.2×~62×, compared to state-of-the-art network trace generator NetShare [47].

This work does not raise any ethical issues.

2 Background and Motivation

In this section, we explain the motivation for network traffic generation and review the limitations of previous work.

2.1 Why Synthetic Traces are Essential

The need for robust network systems drives research, but real trace data are scarce due to privacy concerns [14, 21]. This limits data-driven studies such as traffic classification, behavior analysis, and anomaly detection [46]. Synthetic traces address this issue by enabling experiments while protecting user privacy at the same time.

Overall, synthetic trace generators fall into three categories: (1) simulation-based [2, 26], (2) model-based [31, 37], and (3) machine learning-based [11, 16, 28, 41, 45, 47]. Simulation- and model-based approaches require substantial manual effort and domain expertise, making them difficult to apply in practice. In contrast, ML-based methods learn directly from data to generate high-fidelity traces, making them the most promising direction and the focus of this paper.

2.2 Limited Temporal Pattern Fidelity

Temporal pattern: Temporal pattern describes how the activity or behavior of the network evolves over time, measured through various metrics, including packet rate, throughput, interarrival time between packets, flow duration and so on.

However, we identify significant issues with existing network traffic generators in maintaining temporal pattern fidelity. For instance, as shown in Fig. 2, E-WGAN-GP [28] and STAN [45] completely distort the packet rate, while NetShare [47], the state-of-the-art method, learns a periodic pattern that contradicts reality. These discrepancies can lead

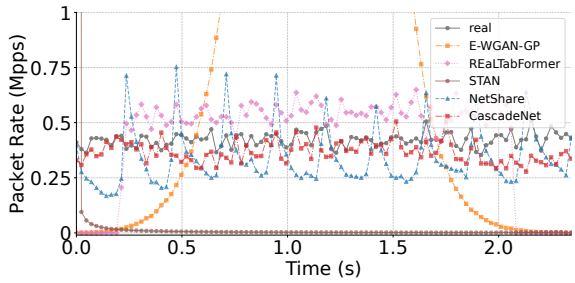


Figure 2: Packet rate of the real trace and synthetic traces generated by various generators for CAIDA dataset.

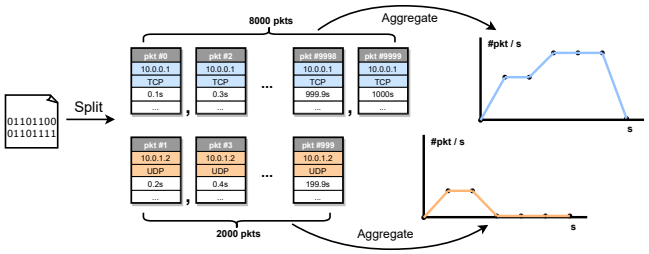


Figure 3: Our approach: Trace as a collection of flows, flows as aggregated time series. (packet rate shown as an example)

to significant deviations in the results of burst analysis and throughput prediction when using synthetic traces, giving rise to misleading conclusions for researchers.

To address this issue, we conduct a thorough analysis of why existing generators struggle to capture temporal patterns and identify that the root cause lies in how the data is represented. Hereby, we can categorize existing ML-based network trace generators into the following types based on their representation approaches:

Approach 1. Trace as a collection of records (Figure 1a): This approach represents a network trace as a collection of independent records. Most early methods followed this representation method, including E-WGAN-GP [28], PAC-GAN [16] and PacketCGAN [41]. Generative models designed for tabular data can also be easily adapted under this representation, for example, CTGAN [44]. Since this representation method entirely overlooks the relationships between records, machine learning models will be unable to capture the temporal patterns in network traces.

Approach 2. Trace as a sequence of records (Figure 1b): Another common way to represent a network trace is as a sequence of records, with STAN [45] serving as a notable example. Generative models for serialized data, including TimeGAN [48], TimeVAE [5], and Doppelganger [15] are also suitable for this representation. Although this method tends to relate all records, it is hard for machine learning models to capture long-term dependencies within the sequence given their auto-regressive nature.

Approach 3. Trace as a collection of flows, flows as sequences of records (Figure 1c): This approach represents a network trace as a collection of flows. [1] NetShare first applied this representation [47], generating each flow's record sequence using time-series GANs. This method not only allows models to produce traces with flow structure but also simplifies learning by focusing on intra-flow relationships.

However, capturing temporal patterns remains challenging. Even though NetShare reduces the sequence lengths by splitting the trace into multiple chunks, elephant flows can still easily exceed tens of thousands of records. Therefore, models still struggle to capture the correct temporal dynamics, as illustrated in Fig.2, showing the inadequacy of this representation to preserve time patterns.

[1] A flow is a sequence of records sharing the same 5-tuple (source IP, source port, destination IP, destination port, protocol).

3 Design Overview

In this section, we first introduce a novel data representation method designed to facilitate the learning of temporal patterns in network traces(§3.1). Next, we discuss the challenges of applying this data representation method in network traffic generation (§3.3). Finally, we present the CascadeNet workflow as an overview of our solution (§3.2).

3.1 Flows as Aggregated Time Series

Our insight: Temporal patterns in network traces are often implicitly embedded as aggregated time series, rather than explicitly represented in record fields like timestamps. For example, packet rate and throughput, defined as the number of packets and bits transmitted over each time interval, serve as aggregated features derived from multiple records. Another commonly considered statistic, interarrival time between records, typically exhibits an inverse relationship with packet rate: the more packets present within the time interval, the shorter their interarrival time.

Therefore, we propose that machine learning models should *explicitly learn temporal patterns from aggregated time series rather than individual records.* We aggregate each flow into a time series and make it one of the training targets for machine learning models. Figure 2 shows the packet rate of the synthetic trace generated by CascadeNet, exemplifying the effectiveness of this representation.

There are two other advantages in aggregating each flow into a time series: (1) **Fidelity:** This approach further simplifies the relationships between records within a flow, preventing the model from becoming entangled in localized patterns. Additionally, the time-series length is determined by the resolution of the aggregated time series, rather than the number of packets in the flow, avoiding the models dealing with the long-term dependencies in time series. This reduces the difficulty

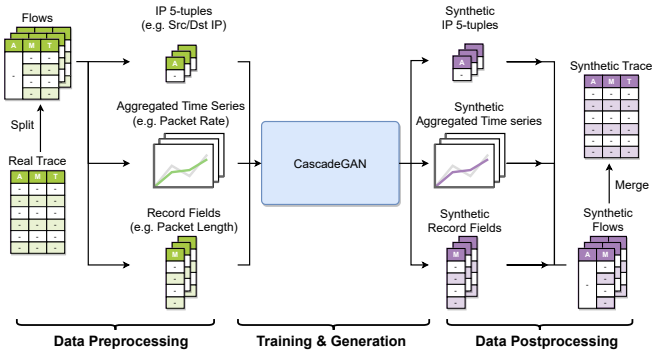


Figure 4: Workflow of CascadeNet. It consists of three stages: Data Preprocessing, Training & Generation, Data Postprocessing.

of the task for the machine learning models, allowing them to achieve better performance and generate synthetic traces with higher fidelity. (2) **Scalability:** This approach streamlines the information and reduces the length of the sequence. This decreases computational complexity for both the training and the generation processes of machine learning models, thus improving scalability. Given these advantages, we propose a novel data representation approach for network traces:

Our approach. Trace as a collection of flows, flows as aggregated time series: We first divide the network trace into different flows according to the IP 5-tuples. After that, we segment each flow into fixed-length time steps and extract key features from each time step to generate an aggregated time series for each flow. Figure 3 illustrates this approach.

3.2 CascadeNet Workflow

We propose CascadeNet, an end-to-end framework for network trace generation based on the novel data representation method. CascadeNet operates in three stages: data preprocessing, training & generation, and data postprocessing, as depicted in Figure 4.

Data preprocessing: First, we split the network trace into flows based on IP 5-tuples and encode them into vectors as unique features for each flow. Specifically, IP addresses are encoded bit by bit to avoid memorization as much as possible, and port numbers and protocols are encoded using Word2Vec [19]. Next, we extract those important features that reflect temporal patterns, including packet rate and throughput, forming an aggregated time series for each flow. Finally, we encode the other fields within the records into vectors, enabling the machine learning model to capture fine-grained features from each record. Categorical fields are encoded into one-hot vectors, and numerical fields are normalized to [0, 1]. In particular, timestamps are ignored at this stage because the models do not learn them directly.

Training and generation: Due to the intrinsic differences between IP 5-tuples, aggregated time series, and record fields, we design a hierarchical model called CascadeGAN to learn these features simultaneously. CascadeGAN follows the standard adversarial learning framework, but incorporates multiple generators and discriminators. During the training phase, the generators and discriminators learn in alternation, leading to discriminators that can effectively differentiate between synthetic and real data and generators capable of deceiving the discriminators. In the generation phase, CascadeGAN's generators produce synthetic IP 5-tuples, aggregated time series, and record fields, followed by the post-processing stage. We will explain the design and implementation details of CascadeGAN in §4.

Data postprocessing: Since machine learning models only produce vector outputs, we need to decode these outputs in the same way they were encoded during the preprocessing stage. After that, we need to reorganize these components into a complete trace. This process involves assigning a timestamp to each record to ensure that the temporal patterns in the complete trace align with the generated time series. We have developed several strategies to achieve this and leave the detailed explanation in §5.1.

3.3 Challenges

There remain several challenges when applying our data representation method in the network traffic generation workflow. The following are the main ones:

Challenge 1: Heterogeneity between features (§4). Our data representation method abstracts network traces into three components: IP 5-tuples, aggregated time series, and record fields. Although both IP 5-tuples and record fields (*e.g.,* IP address and packet length) can be treated as vectors with fixed dimensions, the aggregated time series consists of sequences with varying lengths. There also exists a fundamental distinction between IP 5-tuples and record fields: records in each flow share the same IP 5-tuple, but each record has its unique record fields. These inherent differences in features make it challenging to design a single model capable of learning and generating all of them simultaneously. At the same time, since these features coexist within each record, their interdependencies cannot be ignored, making it impractical to use separate models to generate these features independently.

Challenge 2: Timestamp generation (§5.1). With the novel data representation method, machine learning models no longer rely on timestamps but instead use aggregated time series to capture temporal patterns. Consequently, the models' output will also not include timestamps. A key challenge lies in determining how to generate timestamps that preserve the temporal patterns learned by the model. Inaccurate timestamps can make the temporal patterns of synthetic traces significantly deviate from the generated time series, thereby compromising the ability to achieve high-fidelity synthetic

Dataset	#Flow	#Packet	#Flow/#Packet	%LargeFlow	%LargeFlow-Packet	%Zero	Duration(s)
CAIDA	95,736	998,912	0.096	25.20%	89.12%	88.21%	2.343
DC	49,751	1,000,000	0.050	32.92%	96.00%	53.42%	273.152
CA	539,634	1,000,000	0.540	0.37%	45.26%	25.05%	556.910
TON_IoT	4,294	445,319	0.001	6.15%	98.61%	40.51%	9801.701

Table 1: Statistics about each dataset. **#Flow**: the number of flows; **#Packet**: the number of packets; **#Flow/#Packet**: the ratio of **#Flow** and **#Packet**; **%LargeFlow**: the percentage of flows with ≥ 5 packets; **%LargeFlow-Packet**: the percentage of packets accounted by flows with ≥ 5 packets; **%Zero**: the percentage of zeros in aggregated time series (200 time steps); **Duration**: total duration of the dataset, measured from start to end in seconds.

traces at the last moment.

Challenge 3: Skewed data distribution (§5.2). In our experiments, we found that network traces often exhibit highly skewed distributions, which pose significant challenges for generative models. The first challenge is the *sparsity of the time series*. Table 1 shows that after aggregating the flows into time series, the time series usually contain a large number of zeros. Existing machine learning models struggle to capture the correct distribution in such sparse time series and may even fail to converge during training. The second challenge is the *class imbalance*. Network traces often include a large number of mouse flows and very few elephant flows, which is also illustrated in Table 1. This leads to the issue of mode collapse in generative models, where the model learns only the dominant patterns and neglects the less frequent ones. It is intolerable in our task because elephant flows typically contain most of the packets in the trace: ignoring elephant flows often means a significant drop in fidelity.

4 CascadeGAN Design

In this section, we introduce the hierarchical model CascadeGAN in details. We first formulate the problem in §4.1. Then, we briefly explain what are GANs and why we choose GANs in §4.2. We present the detailed design of CascadeGAN in §4.3 and how to effectively train the model in §4.4.

4.1 Problem Formulation

We are given an IPv4 packet-level header trace as input. First, the trace is divided into flows based on IP 5-tuples, denoted as $\{A_i \mid i = 1,2,\ldots,N\}$, where N is the total number of flows. Each flow is then aggregated into a time series, represented as $\{S_i \mid i = 1,2,\ldots,N\}$, where $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,T})$ and T is the length of the time series. Finally, we extract the record fields, denoted as $\{M_{i,t,k} \mid i = 1,2,\ldots,N; t = 1,2,\ldots,T; k = 1,2,\ldots,K_{i,t}\}$, where $K_{i,t}$ represents the number of records in flow i at time step t.

Our objective is to train a generative model G to produce synthetic IP 5-tuples, aggregated time series, and record fields, denoted as $(\{\tilde{A}_i\}, \{\tilde{S}_i\}, \{\tilde{M}_{i,t,k}\})$. We aim for the joint distribution of these synthetic features to closely approximate the real distribution. If this goal is achieved, we can expect the post-processed synthetic traces to closely match real traces in both statistical and temporal patterns. The synthetic traces can also sustain consistent performance in downstream tasks.

4.2 Why GANs

Generative Adversarial Networks (GANs) [7] have shown impressive ability in generating realistic and high-quality synthetic data. A GAN consists of a generator and a discriminator: the discriminator learns to distinguish real from generated samples, while the generator learns to deceive the discriminator. Through alternating optimization, the generator eventually produces data that approximate the real distribution.

Extensive studies have explored applying GANs to tabular and time-series data generation [34], as well as to network trace synthesis [16, 28, 41, 47]. Although Transformer [39]- and Diffusion-based [9, 35, 38] generative models have recently gained attention, their substantial computational and data requirements make them less practical for this task.

4.3 CascadeGAN Structure

The novel data representation method we proposed abstracts network traces into three heterogeneous components: IP 5-tuples, aggregated time series and record fields. Each of these components captures different levels of granularity in the network trace: IP 5-tuples act as identifiers for each flow, representing global characteristics at the coarsest granularity; aggregated time series describe the dynamics of each flow over time, offering a more detailed view; and record fields provide the finest granularity, containing all information for each flow at each time step.

To leverage this structure, we design CascadeGAN, a hierarchical model that progressively learns the various features embedded in network traces from coarse-grained to fine-grained, as shown in Fig.5. CascadeGAN is composed of three generators and three discriminators, trained adversarially in pairs to learn the distributions of IP 5-tuples, aggregated time series and record fields respectively. To capture the relationships between these components, we convert the task of learning the

joint distribution, which is challenging due to the heterogeneity of the components, into learning the equivalent conditional distributions. Specifically, Generator 2 and Discriminator 2 take IP 5-tuples as input, while Generator 3 and Discriminator 3 take both IP 5-tuples and aggregated time series as input. The hierarchical structure, from coarse-grained to fine-grained, minimizes the model's computational complexity: upper-layer models appear earlier in the forward process and later in the backward process, leading to more gradient calculations compared to lower-layer models. Following is the detailed explanation for each generator and discriminator:

Generator 1: IP 5-tuples generation. Generator 1 generates the IP 5-tuple fields $\tilde{A}_i = G^{(1)}(z_i^{(1)})$, where $z_i^{(1)}$ is a noise vector. We choose Multi-Layer Perceptron(MLP) [30] as the backbone, which is widely used in learning tabular data.

Generator 2: Aggregated time series generation. Generator 2 generates the aggregated time series $\tilde{S}_i = G^{(2)}(\tilde{A}_i, z_i^{(2)})$, where $z_i^{(2)}$ is a noise vector. It aims at capturing the flow-level temporal pattern within the real trace, and the input $\tilde{A}_i$ enables Generator 2 to generate unique characteristics for different flows. Therefore, we choose Long Short-Term Memory(LSTM) [10] as the backbone for its strong ability to understand sequential data. Additionally, experiments have also demonstrated that LSTM is a better choice for Generator 2 than MLP (Appendix E).

Generator 3: Record fields generation. Generator 3 generates the record fields $\tilde{M}_{i,t,k} = G^{(3)}(\tilde{A}_i, wnd(\tilde{S}_i, t), z_{i,t,k}^{(3)})$, where $z_{i,t,k}^{(3)}$ is a noise vector, and $wnd(\tilde{S}_i, t)$ denotes a sliding window function that extracts a fixed-size window around the time step t from the aggregated time series $\tilde{S}_i$. The sliding window design enables Generator 3 to capture sufficient temporal features to generate corresponding record fields, while avoiding the inclusion of excessive information which could be a heavy burden. There are two options for backbone of Generator 3: MLP and LSTM. Although LSTM can theoretically learn the correlations between different records within the same time step, the number of records at each time step remains difficult to control. This leads to issues similar to those in NetShare (§2.2). In contrast, the MLP architecture aligns better with our design principles: capturing temporal patterns through aggregated time series and simplifying the correlations between records within the same time step. Experiments also support the choice (Appendix E).

CascadeGAN also includes three discriminators $D^{(i)}, i = 1,2,3$, corresponding to the generators, all of which adopt the MLP structure. In the adversarial training, we use the loss function from Wasserstein GAN with Gradient Penalty (WGAN-GP) [8], one of the most popular variants of GANs. The formulas are shown in Appendix C.

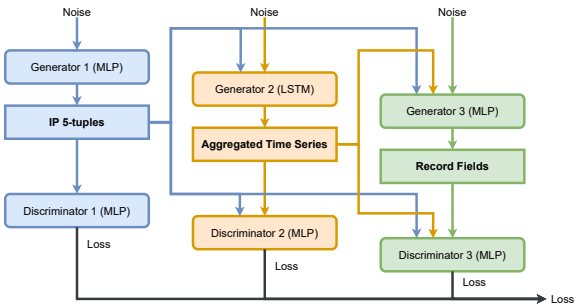


Figure 5: Architecture of CascadeGAN.

4.4 Hierarchical Model Training

GANs are notorious for their difficulty in training, and the hierarchical nature of CasacadeGAN makes it particularly noteworthy to mention here. This is because both the forward and backward processes in the training need to go through all generators and discriminators. Especially during the backward process, gradients keep accumulating, and the computational complexity sky-rockets. The hierarchical structure also causes training issues that we call "cascading divergence". Before Generator 1 is well trained, it could mislead the training of Generator 2 by passing a distorted input. Similarly, the distorted result from Generator 2 can also mislead the training of Generator 3.

To solve this issue, we adopt a two-phase training strategy, including a separate pretraining phase followed by a combined finetuning phase. In the pretraining phase, we train each pair of generator and discriminator independently using their respective loss functions $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_3$ to ensure focused and effective learning of their specific tasks. Instead of synthetic IP 5-tuples and aggregated time series, Generator 2 and Generator 3 take the real features as their input during this phase. In the finetuning phase, we train all generators and discriminators together using a combined loss function $\mathcal{L} = \alpha\mathcal{L}_1 + \beta\mathcal{L}_2 + \mathcal{L}_3$, where α and β are weighting factors that balance the contribution of each generator's loss to the overall training process.

The two-phase training strategy not only boosts the overall performance but also gives us a more fine-grained control to the training process. While GAN1 and GAN2 are given flows as samples, GAN3 is given packets as samples. As demonstrated in Table 1, the flow-to-packet ratio varies substantially between datasets. Given such drastic variation, it is impractical to balance the training across all generators and discriminators if they are trained together from the beginning. With the two-phase approach, we can tailor the training regimen to the specific characteristics of each dataset.

5 Timestamp Generation and Optimizations

In this section, we first introduce the timestamp generation techniques in the postprocessing stage (§5.1). After that, we present several machine learning optimization techniques to address the issue of skewed data distribution in network traces (§5.2).

5.1 Generating Timestamps with High Fidelity

Unlike most machine learning-based network traffic generators, CascadeNet learns temporal patterns from aggregated time series rather than relying on timestamps. As a result, it does not directly generate timestamps, which aligns with its design principles. To address this limitation, we develop several timestamp generation techniques that allow CascadeNet to produce complete, high-fidelity synthetic traces:

Equidistant generation (EQ): The core idea is to generate timestamps evenly within each flow and time step, ensuring the synthetic trace aligns with the model's aggregated time series. However, it fails to capture instantaneous dynamics within each step, potentially reducing fidelity.

Sub-period equidistant generation (SP): This technique stems from the observation that records tend to cluster within a shorter span of each time step. We therefore introduce a new aggregated dimension, the sub-period, defined as the interval between the earliest and latest records within a step. The model learns this feature along with others, and timestamps are later assigned evenly within each sub-period. This addition enables finer temporal modeling and improves the fidelity of synthetic traces.

ML-based generation (ML): This technique follows the concept of using machine learning for data generation. During preprocessing, we incorporate each record's relative timestamp within its time step into the record fields, allowing GAN 3 to learn and generate these relative timestamps. In the postprocessing step, each record's timestamp is reconstructed from its relative timestamp based on the time step it belongs to. Since records are confined within their respective time steps, this approach also ensures that the temporal patterns of the synthetic trace strictly follow the aggregated time series produced by the model. Moreover, the integration of machine learning models offers the potential for synthetic traces to replicate more detailed temporal patterns.

Appendix F shows the performance of the timestamp generation techniques on temporal pattern fidelity. We use Earth Mover's Distance (EMD) to measure the difference between real and synthetic traces across various metrics, as detailed in §7.2. The results indicate that the ML-based generation technique can produce synthetic traces with high-fidelity temporal patterns in most cases. Consequently, we will use this technique by default for generating timestamps in subsequent analyses.

5.2 Optimization Techniques

During the implementation of CascadeNet, we observe that the skewed data distribution in network traces negatively impact the performance of generative models and could even lead to convergence issues during training. To address these challenges, we employ a series of optimization techniques to ensure that the generative model performs well in different cases.

Zero inflation method. We observed that the aggregated time series generated during the preprocessing stage often contain a significant amount of zeros. This sparsity makes it challenging for the generator to produce synthetic aggregated time series with similar distributions, leading to an imbalance between the discriminator and the generator.

To address this problem, we employ the zero-inflation method, inspired by Lambert's seminal work [13]. A Zero-inflated model consists of two parts: a zero-inflated component dominating whether the value should be zero, and a non-zero inflated component depicting the distribution of non-zero values. Adapting this approach to our context involves introducing an additional feature in the aggregated time series to signal the presence or absence of activity in each time step. This optimization allows the generator to explicitly account for data sparsity, differentiating between time steps with or without packet transmission. Notice that this technique is specific for CascadeNet since network traffic data with other representation methods is usually dense enough.

Conditional input strategy. In network traces, there are typically a few elephant flows and a large number of mouse flows. This imbalance often leads models to completely disregard the features of the elephant flows and only focus on the mouse flows, which is called mode collapse [7]. Given that the elephant flows contain a significant amount of records, the mode collapse can severely undermine the fidelity of the synthetic trace. Common methods to address this issue include resampling and data augmentation, but they are not suitable for our task as they alter the data distribution of the original trace.

To mitigate the issue, we introduce several flow-level features as conditional inputs to the generators, including *flow duration*, *number of packets per flow*, *maximum packet rate*, *etc.* The additional flow-level features act as contextual cues, avoiding interleaving the learning processes between different types of flows. Notice that DoppelGANger [15], the backbone model of NetShare, utilize another technique that normalizes each time series individually to tackle mode collapse. However, in our initial experiments, we found that it was not effective for our model.

Furthermore, this approach introduces a novel possibility. If desired, users can reconfigure the input features to manually adjust the proportions of different types of flows, thereby better tailoring the model to specific use cases. Although this form of customizable data sharing presents a promising direction, we leave its deeper exploration for future work.

We conducted a comprehensive ablation study on these two optimization techniques, as shown in Appendix G. The evaluation demonstrates that both techniques significantly enhance the temporal pattern fidelity of CascadeNet, effectively mitigating issues of data sparsity and mode collapse. For the conditional input strategy, we also conducted an ablation study on different input features. The results indicate that *flow duration* and *maximum packet rate* are the most influential features, while the remainings affect various temporal pattern metrics to differing extents. Depending on the use case, users may choose to include or exclude certain features.

6 Implementation

We implement the prototype of CascadeNet with Pytorch 2.1.0 [24], comprising over 4000 lines of code. The source code is available at https://github.com/lurw2000/CascadeNet. For consistent measurement of runtime, all experiments are run with same configuration on a workstation with 64 CPU cores, 512GB RAM and 4 NVIDIA GeForce RTX 4090. Each experiment is allocated 8 CPU cores and 1 GPU.

We pre-define a set of hyperparameters for CascadeNet in a JSON file. These include hidden layer sizes, noise dimensions, batch size, the number of epochs for training, and so on. Before conducting our evaluation, we examined several options for each hyperparameter and selected the best one according to the distance metrics of different features in the trace, as shown in Appendix D. We find that CascadeNet is not sensitive to most hyperparameters, but the length of the aggregated time series is an important variable, which will be further discussed in §9.

For evaluation, we use SciPy 1.10.1 to calculate JSD and EMD [40]. We use NOLDS, a Python library that implements nonlinear measures for dynamical systems, to calculate the hurst exponent of aggregated time series [31]. For downstreaming tasks, we use NetML [46] to perform anomaly detection, and use Scikit-learn 1.2.2 [25] and Statsmodels 0.14.0 [32] to implement different prediction models.

7 Evaluation

In this section, we evaluate the performance of CascadeNet and compare it with existing ML-based network trace generators. We first describe the experimental setup, including the datasets and baselines (§7.1). We then answer the following key questions:

- *Does CascadeNet preserve the statistical and temporal patterns of the real traces? (§7.2)* By measuring the distance between synthetic traces and real traces, we find CascadeNet achieves 41%~76% better accuracy on temporal distance metrics and comparable accuracy on statistical distance metrics.
- *Could the synthetic traces generated by CascadeNet replace the real traces in downstream tasks? (§7.3)* In the tasks of burst analysis, throughput prediction and anomaly detection, the synthetic traces generated by CascadeNet produce results much closer to the real cases compared to those produced by the baselines.
- *Does CascadeNet have sufficient diversity to generate samples that are different from the training data?* We examined the synthetic traces of CascadeNet and found that it rarely generates the same samples as those in the training data.
- *How efficiently does CascadeNet perform and scale up to large traces? (§7.5)* Compared to the state-of-art approach NetShare, CascadeNet reduce the training and generating time for 7.3×~25× and 9.2×~62×.

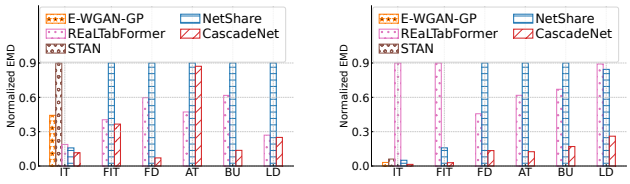
7.1 Setup

Datasets: To guarantee the generalizability of our evaluation results, we use four pcap traces collected from various networks. For traces larger than 1 million records, we select a subset of 1 million consecutive records for consistency.

- *CAIDA* [1]: This is a packet capture from a backbone link of a tier-1 ISP. We use the subset of trace from the New York collector in March 2018.
- *DC* [3]: This is a packet capture from a university data center. It is called "UN1" in the paper.
- *CA* [23]: This is a trace from the U.S. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competitions held in March 2012.
- *TON_IoT* [22]: This is a packet capture from telemetry IoT sensors. We use the raw network datasets that contain only normal traffic.

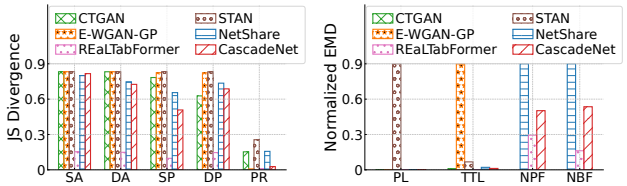
Baselines: We compare CascadeNet to the latest ML-based network traffic generators and a GAN-based model design for tabular data [44].

- *CTGAN* [44]: CTGAN is the state-of-the-art GAN-based generator for tabular data. While it is not designed for network traffic data, we extend it by encoding all fields into categorical or numerical variables.
- *E-WGAN-GP* [28]: E-WGAN-GP is a GAN-based generator for network traffic data. It uses IP2Vec [27] to embed each record into a vector and applies WGAN-GP [8] to generate synthetic records. We extend the original E-WGAN-GP to accommodate IPv4 header traces.
- *STAN* [45]: STAN is an auto-regressive-based generator. It initiates traffic generation by sampling records from the marginal distributions, then continuously generates new records based on prior ones in an auto-regressive fashion. We extend the original STAN to support IPv4 header traces.
- *NetShare* [47]: NetShare is the state-of-the-art GAN-based network traffic generator. Netshare first generates the IP

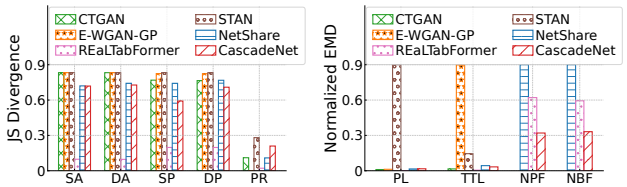


(a) CAIDA (b) DC

Figure 6: Normalized EMD (↓) between the temporal patterns of real traces and synthetic traces.



(a) CAIDA



(b) DC

Figure 7: JSD (↓) and Normalized EMD (↓) between the statistical patterns of real traces and synthetic traces.

5-tuple for each flow, and then sequentially generates the other fields for each packet within the flow.

- *NetDiffusion* [11]: NetDiffusion is a diffusion-based generator. NetDiffsuion first converts each flow into an image, then uses models finetuned from Stable Diffusion 1.5 [29] to generate synthetic images.
- *REaLTabFormer* [36] REaLTabFormer is a framework for synthesizing tabular data of different types using fine-tuned GPT-2. We extend it to fit network traffic data.

Time granularity: As the trace durations differ significantly (Table 1), we do not fix a specific time granularity when evaluating temporal patterns. Instead, each trace is divided into sequences of identical length, so that longer traces correspond to coarser time resolutions and shorter traces to finer ones, enabling a fair comparison of CascadeNet with baselines across different temporal granularities.

Notably, we could not train STAN on the TON_IoT dataset, as it consistently crashed with various parameter settings. Similarly, NetDiffusion requires weeks to months to generate traces on the CAIDA, CA, and DC datasets due to their large flow volumes. We consequently excluded these results.

7.2 Fidelity

First, we evaluate how CascadeNet preserves the statistical and temporal patterns of real traces.

Temporal pattern fidelity. We measure the distance between real and synthetic traces on the following temporal features: (1) *Packet-Level Interarrival Time(IT)* : The interarrival time between records; (2) *Flow-Level Interarrival Time(FIT)*: The interarrival time between records within each flow; (3) *Flow Duration(FD)*: The difference between the timestamp of the last and first record within each flow. (4) *Average Throughput(AT)*: The average throughput of each flow; (5) *Burstiness(BU)*: Measured by the coefficient of variation of each flow's packet rate; (6) *Long-term Dependency(LD)*: Measured by the Hurst exponent of each flow's packet rate, which is an important property of complex network traffic [4, 42]. For metrics AT, BU, and LD, each flow's packet rate is estimated by dividing into 400 uniform time intervals.

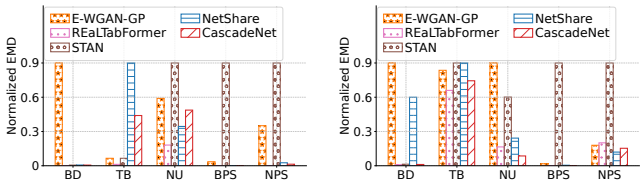
We choose the Earth Mover's Distance (EMD) as the distance metric, and normalized the distance to $[0, 0.9]$ for better visualization. Fig.6 shows the result. CTGAN and NetDiffusion are excluded because they do not generate timestamps. E-WGAN-GP and STAN only have the IT metric since they can not generate traces with flow structure. Compared to baselines except REaLTabFormer, CascadeNet achieves 41%~76% improvement in temporal pattern fidelity. Although REaLTabFormer demonstrates reasonably good fidelity in capturing temporal patterns, it primarily relies on memorizing and reproducing content from the training data. We will elaborate on this in §7.4.

Statistical pattern fidelity. We measure the distance between real and synthetic traces on following statistical features: *Source Address* (SA), *Destination Address* (DA), *Source Port* (SP), *Destination Port* (DP), *Protocol* (PR), *Packet Length* (PL), *Time to Live* (TTL), *Number of Packets per Flow* (NPF) and *Number of Bytes per Flow* (NBF). For numerical features including PL, TTL, PF and BF, we choose EMD as the distance metrics; for categorical features including SA, DA, SP, DP and PR, we choose Jenson-Shannon Divergence (JSD) as the distance metrics. For better visualization, we normalized the EMD to $[0, 0.9]$.

Figure 7 shows the JSD and normalized EMD between real and synthetic traces. CTGAN, E-WGAN-GP and STAN do not have the PF and BF metrics since they can not generate traces with flow structure. The result shows that CascadeNet outperforms most baselines except REaLTabFormer. We again emphasize that the high fidelity of REaLTabFormer comes largely from the memorization of the training data (§7.4).

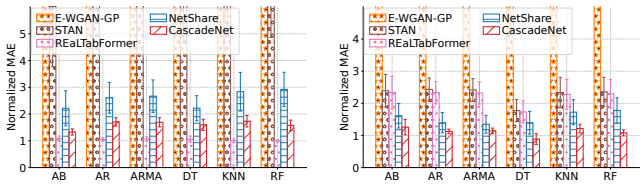
7.3 Downstream Tasks

Next, we evaluate whether CascadeNet-generated traces could represent the real traces on downstream tasks. In previous studies [11, 28, 45, 47], existing trace generators have already



(a) CAIDA (b) DC

Figure 8: Normalized EMD (↓) between burst analysis metrics of real traces and synthetic traces.



(a) CAIDA (b) DC

Figure 9: Normalized MAE (↓) between real throughput and the prediction by different ML-models trained on synthetic throughput.

demonstrated a high degree of consistency with real data in downstream tasks that are not sensitive to temporal patterns. However, since the insufficient fidelity in temporal pattern observed, here we focus on evaluating downstream tasks that are highly dependent on temporal patterns: (1) burst analysis; (2) throughput prediction; (3) ML-based anomaly detection.

Burst analysis. Bursts have emerged as frequently discussed issues in network performance. We follow a recent study [49] to analyze differences in the bursts observed in real-world traces compared to synthetic traces. To accommodate the heterogeneous durations of datasets, all traces are divided into 1000 uniform time intervals, and bursts are defined as maximal sequences of intervals with utilization over 0.5. Bursts are examined from the following perspectives: (1) BD: Burst Durations; (2) TB: Time between Bursts; (3) NU: Network Utilization; (4) BPS: Burst Packet Sizes; (5) NPS: Non-burst Packet Sizes. We perform burst analysis on both real and synthetic traces, and use normalized EMD to measure their differences. Fig.8 shows the result. Excluding REaLTabFormer, CascadeNet achieves either the best or near-best performance across all metrics. CTGAN and NetDiffusion are also excluded because they do not generate timestamp.

Throughput prediction. Throughput prediction is also an area of focus. We choose six models for evaluation: Autoregression(AR), Autoregressive moving average(ARMA), K-Nearest Neighbors(KNN), Decision Tree(DT), Random Forest(RF), AdaBoost(AB). First, we divided both the real and synthetic traces into 200 uniform time intervals to obtain the throughput time series. Then we use the first 10%, 30%, 50%, 70% and 90% of data to train the prediction models and

Generator	CAIDA	DC	CA	TON_IoT
CTGAN	100%	100%	100%	100%
E-WGAN-GP	100%	100%	100%	100%
STAN	100%	100%	100%	/
NetShare	100%	100%	100%	99.9%
NetDiffusion	/	/	/	100%
REaLTabFormer	11.4%	6.9%	56.5%	1.7%
CascadeNet	100%	99.9%	99.9%	99.9%

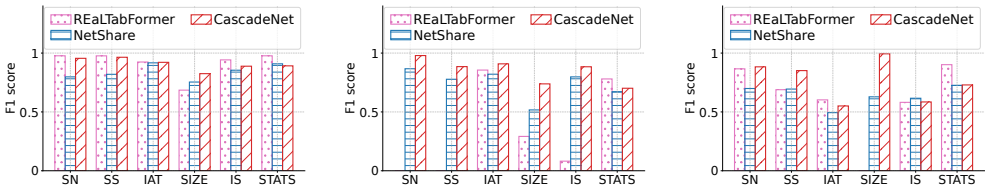
Table 2: Proportion of records (↑) in the synthetic traces that differ from the training data.

make them predict the latter 10% of data. We measure the prediction accuracy by normalized mean absolute error(MAE) MAE_{syn}/MAE_{raw}, where MAE_{raw} refers to MAE of the real trace and MAE_{syn} refers to MAE of the synthetic trace. Fig.9 shows that CascadeNet outperforms the baselines in most cases, excluding REaLTabFormer. Again, CTGAN and NetDiffusion are excluded because they do not generate timestamps.

ML-based anomaly detection. Network anomaly detection has emerged as a highly dynamic and influential research area. We choose a recent open source library NetML [46] for evaluation. NetML first characterizes all flows as different representations(also called models): (1)interarrival time(IAT); (2) packet size(SIZE); (3) interarrival time and packet size(IS); (4) a series of statistics including duration, packet rate, *etc.* (STATS); (5) number of packets sampled by sub-windows (SN); (6) number of bytes sampled by sub-window (SS). Subsequently, it detects anomalous flows using a one-class support vector machine (OCSVM). To evaluate the fidelity of synthetic traces, we run NetML on real traces and synthetic traces separately, and compare their result on real traces by calculating the F_1 score. Fig.10 shows that CascadeNet keeps higher accuracy than NetShare across most models and datasets with few exceptions. Baselines other than NetShare and REaLTabFormer are excluded because they do not generate timestamps or do not generate traces with flow structure. The dataset TON_IoT is also excluded since it does not contain anomalous flows.

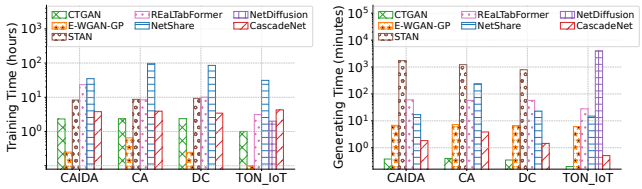
7.4 Diversity

For generative models, another noteworthy metric lies in the diversity of their outputs. A generative model is inadequate if it can only memorize the content of the training data but does not generate output that differs from it. We calculated the proportion of records in the synthetic traces that are exactly the same as those in the training data, as shown in Table 2. Like most of the baselines, CascadeNet is capable of generating highly diverse inputs. However, REaLTabFormer, as a representative of LLM-based models, exhibits a severe lack of diversity. The key issue here is that, compared to models like GANs and Diffusion models, transformer-based architectures



(a) CAIDA (b) DC (c) CA

Figure 10: F1-score (↑) of the prediction by anomaly detection models trained on synthetic traces.



(a) Training Time (b) Generating Time

Figure 11: Training and generating time (↓) of different network traffic generators on various datasets.

do not incorporate enough noise in their inputs, making them more prone to memorizing data rather than truly generating new ones. Only by targeting a general model and using large-scale data for training can these models become diverse.

7.5 Scalability

Next, we measure the training and generating time of different ML-based trace generators, as shown in Fig.11. CTGAN and E-WGAN-GP exhibit the shortest training and generation times, which is expected, as these models do not account for relationships between records. NetDiffusion's training process is also relatively fast, as it only requires fine-tuning a pre-trained model rather than training from scratch. However, its approach of converting each flow into an image significantly increases the data generation time, making the process seem endless when dealing with datasets containing a large number of flows, such as CAIDA, CA, and DC. In comparison to NetShare, a state-of-the-art method, CascadeNet reduces training time by 7.3×~25× and generation time by 9.2×~62×.

8 Related Works

Network simulators. Network simulation [2, 26] could be a method to generate synthetic network traces. However, network simulators often require configurations with extensive parameters. As a result, using network simulation to obtain synthetic network traces can be time-consuming and labor-intensive, and it is also hard to generalize across different scenarios.

Mathematical model-based generators. Mathematical model-based generators, for example, Harpoon [37] and Swing [31], use stochastic or statistical models to capture network activities. These methods are more practical compared to network simulations, but the underlying mathematical assumptions limit their generalizability.

Machine learning-based generators. Capitalizing on the latest advances in machine learning, these generators employ ML techniques to learn from actual network data and generate synthetic traces [11, 16, 28, 41, 45, 47]. ML-based methods are particularly promising, offering the potential to produce synthetic traces that closely mirror the complex characteristics of real network traffic. However, current methods generally face problems with temporal pattern fidelity, which becomes the motivation for this work.

GAN-based models. Since the advent of adversarial learning [7], considerable efforts have been dedicated to extending and improving the vanilla GAN, including WGAN-GP [8], RCGAN [6], TimeGAN [48], Doppelganger [15], *etc.* However, due to our unique data representation for network traces, these models cannot be directly applied to our workflow. We therefore design CascadeGAN to overcome this challenge.

Other generative models. Generative models based on transformers [39] and diffusion models [9,35,38] have attracted significant attention. Transformers, leveraging attention mechanisms, excel in context understanding and have become nearly the exclusive solution in fields like natural lauguage processing. Diffusion models have also emerged as the leading choice in image generation, surpassing the dominance of VAEs [12] and GANs [7]. However, our evaluation shows that directly applying these models to network traffic generation does not lead to improved performance.

9 Discussion and Future Work

Although CascadeNet significantly improves the fidelity and efficiency of network traffic generation, it does not address all problems. We will discuss the limitations of CascadeNet and suggest future directions for improvement.

Choice of Time Series Length The time series length is a critical parameter for CascadeNet. As analyzed in §2.2, excessively long sequences complicate training, while overly

short ones weaken representation and reduce temporal fidelity. Appendix H shows that when the length is large (≥ 500), the model's fitting capability drops significantly. In contrast, even with short sequences of 20, the generated traces still maintain adequate fidelity, owing to the timestamp recovery technique. Hence, it is advisable to use a relatively short time series length (≤ 200) for CascadeNet.

Consistency in performance ranking Prior evaluations primarily assess how models trained on synthetic traces perform when applied to real traces, which reflects the utility of synthetic data for downstream deployment. However, when synthetic traces are used for model development rather than application, a stronger requirement emerges: the performance ranking on synthetic traces should be consistent with the ranking on real traces. While such consistency may naturally arise as the fidelity of synthetic traces improves, developing generation methods that explicitly preserve model ranking consistency represents an important research direction.

Payload data CascadeNet currently does not generate payloads. One potential solution is to incorporate an additional layer into CascadeNet's hierarchical model specifically designed for payload generation. However, unlike header fields, the payloads may contain more complex information, such as application layer protocols and natural languages, making this task particularly challenging.

Generalize capability CascadeNet, like other network traffic generators trained from scratch, may have problems with generalization. In practice, the training traces may suffer from issues like insufficient data or skewed distributions, which can limit the model's performance. A potential solution is to pre-train a general model using large numbers of traces. Even if the target trace is of lower quality, fine-tuning a general model with it can still produce adequate results.

Extending to other use cases Although CascadeNet has significantly broadened its applications by supporting downstream tasks that are sensitive to temporal patterns, it still faces challenges in scenarios that heavily depend on inter-flow interactions—for example, evaluating congestion control protocols or understanding how end applications respond to network failures. Given the sheer number of flows, using one single model to capture all inter-flow relationships is impractical. Therefore it is crucial to obtain a dependency graph that captures relationships between flows. While it is possible for expertise to construct such a graph manually, a promising future direction is to leverage machine learning models to infer these relationships automatically from network traces.

Differential Privacy Privacy issues surrounding synthetic traces are a significant concern. While the goal of differential privacy is highly attractive, previous research [47] has demonstrated that even weak differential privacy introduced during machine learning training can severely compromise the fidelity of synthetic traces. Furthermore, we find that differential privacy training for adversarial models using PyTorch, one of the most popular frameworks for deep learning, results in $100\times$ speed reduction. These factors together suggest that differential privacy may not be the most effective solution.

Personally Identifiable Information(PII) In a typical IPv4 header trace record, the only PIIs involved are the source and destination IPs. Existing works [20, 33, 43] propose various approaches to anonymize network IPs without impacting utility properties such as network prefixes, which are widely used in the publicly available network trace datasets. These anonymization techniques are orthogonal to our work. They can be additionally applied to our synthetic trace to further guarantee IP privacy without modifying the CascadeNet workflow or impacting the fidelity of the output trace.

Implicit System and Application Information Leakage: A network trace might implicitly leak system or application-related network information, from statistics such as the number of IP prefixes and flows, or the number and length of the packets per flow. In CascadeNet, the number of flows in a trace is configurable, which also controls the number of distinct IPs in the trace, eliminating privacy concerns of system scale or network topology leakage. However, arbitrarily modifying the number of flows may reduce fidelity, and users should carefully balance fidelity and privacy according to the requirements of downstream tasks. On the other hand, the number and length of the packets per flow are learned and generated by the GAN models, which intrinsically provide some level of data diversity and generalizability. We recognize that with deep analysis of the distribution of the total or individual packet length per flow, it might be possible to deduce the type of application involved in a trace. We do not provide privacy guarantees in these aspects and leave these for future works.

10 Conclusion

In this paper, we propose an end-to-end framework, CascadeNet, that generates network traffic with high-fidelity temporal patterns. Our evaluation shows that CascadeNet outperforms the state-of-the-art models on both fidelity and scalability, and the synthetic traces generated by CascadeNet have better consistency with real traces on downstream tasks.

Acknowledgments

We thank our shepherd Kevin Hsieh and anonymous reviewers for their constructive feedback. This work is supported by National Natural Science Fund for the Excellent Young Scientists Fund Program (Overseas) and Peking University startup fund. Guyue Liu is the corresponding author.

References

[1] The CAIDA UCSD anonymized internet traces - passive collector: equinix-nyc in march 2018.

[2] The network simulator - ns-2.

[3] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM.

[4] M.E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. 5(6):835–846.

[5] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation.

[6] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans.

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein GANs.

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. 9(8):1735–1780.

[11] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. NetDiffusion: Network data augmentation through protocol-constrained traffic generation. 8(1).

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes.

[13] Diane Lambert. Zero-inflated poisson regression, with an application to defects in manufacturing. 34(1):1–14.

[14] Liang Guo and I. Matta. The war between mice and elephants. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 180–188. IEEE Comput. Soc.

[15] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using GANs for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483.

[16] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. PacGAN: The power of two samples in generative adversarial networks.

[17] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 334–350. Association for Computing Machinery.

[18] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 101–114. Association for Computing Machinery.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.

[20] Greg Minshall. TCPdpriv. 1997. https://ita.ee.lbl.gov/html/contrib/tcpdpriv.html.

[21] T. Mori, R. Kawahara, S. Naito, and S. Goto. On the characteristics of internet traffic variability: spikes and elephants. In *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, pages 99–106. IEEE Comput. Soc.

[22] Nour Moustafa. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_iot datasets. 72:102994.

[23] Netresec. MACCDC dataset.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. 12:2825–2830.

[26] George F. Riley and Thomas R. Henderson. The ns-3 network simulator. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 15–34. Springer Berlin Heidelberg.

[27] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. IP2vec: Learning similarities between IP addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 657–666.

[28] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. 82:156–172.

[29] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models.

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. 323(6088):533–536.

[31] Christopher Schölzel. Nonlinear measures for dynamical systems.

[32] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

[33] Adam Slagell, Jun Wang, and William Yurcik. Network Log Anonymization: Application of Crypto-Pan to Cisco Netflows. In *Proceedings of the Workshop on Secure Knowledge Management 2004*, September 2004.

[34] Kaleb E. Smith and Anthony O. Smith. Conditional GAN for timeseries generation.

[35] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265. PMLR.

[36] Aivin V. Solatorio and Olivier Dupriez. Realtabformer: Generating realistic relational and tabular data using transformers. *arXiv preprint arXiv:2302.02041*, 2023.

[37] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: A flow-level traffic generator for router and network tests. 32(1):392.

[38] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[40] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. 17:261–272.

[41] Pan Wang, Shuhang Li, Feng Ye, Zixuan Wang, and Moxuan Zhang. PacketCGAN: Exploratory study of class imbalance for encrypted traffic classification using CGAN. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

[42] W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. 5(1):71–86.

[43] Jun Xu, Jinliang Fan, Mostafa H. Ammar, and Sue B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289, Paris, France, 2002. IEEE Comput. Soc.

[44] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *Advances in Neural Information Processing Systems*.

[45] Shengzhe Xu, Manish Marwah, Martin Arlitt, and Naren Ramakrishnan. STAN: Synthetic network traffic generation with generative neural models.

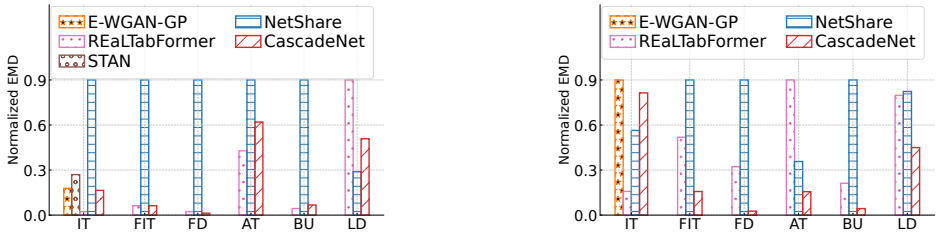
[46] Kun Yang, Samory Kpotufe, and Nick Feamster. Feature extraction for novelty detection in network traffic.

[47] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical GAN-based synthetic IP header trace generation using NetShare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 458–472. ACM.

[48] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

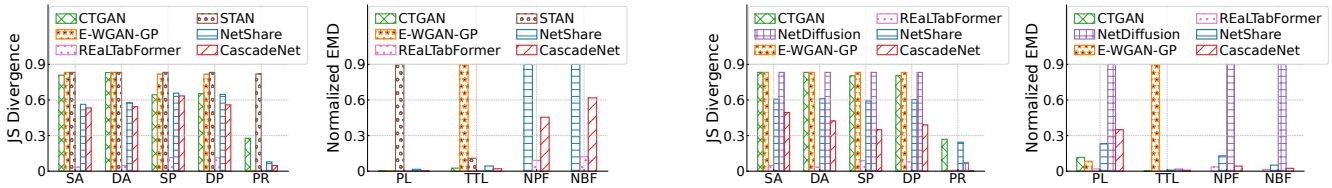
[49] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, pages 78–85. Association for Computing Machinery.

A Additional Fidelity Results



(a) CA (b) TON_IoT

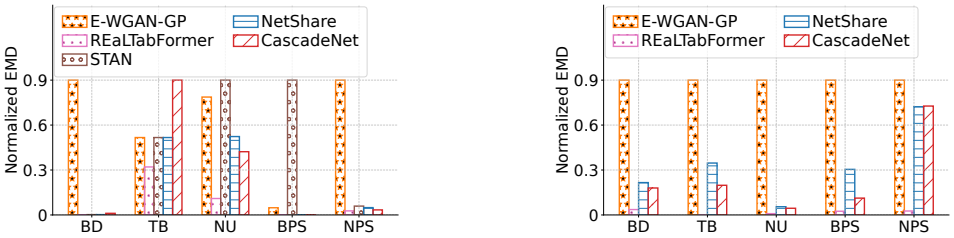
Figure 12: Normalized EMD (↓) between the temporal patterns of real traces and synthetic traces.



(a) CA (b) TON_IoT

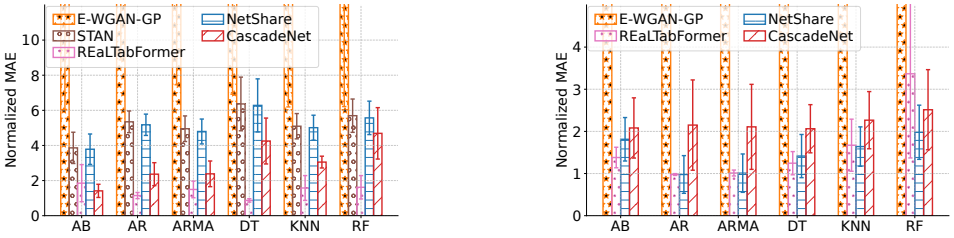
Figure 13: JSD (↓) and Normalized EMD (↓) between the statistical patterns of real traces and synthetic traces.

B Additional Downstream Tasks Results



(a) CA (b) TON_IoT

Figure 14: Normalized EMD (↓) between micro-burst analysis metrics of real traces and synthetic traces.



(a) CA (b) TON_IoT

Figure 15: Normalized MAE (↓) between real throughput and the prediction by different ML-models trained on synthetic throughput.

C Loss functions of CascadeGAN

The loss functions of GAN 1,2 &3 are

$$\mathcal{L}_1 = \mathbb{E}[D^{(1)}(\tilde{A})] - \mathbb{E}[D^{(1)}(A)] + \lambda_1 \mathbb{E}[(\|\nabla D^{(1)}(\hat{A})\| - 1)^2]$$

where A is the IP 5-tuple of the real trace, $\tilde{A}$ is the IP 5-tuple of the synthetic trace, λ_1 is the gradient penalty coefficient, and $\hat{A}$ is the random interpolation of $\tilde{A}$ and A.

$$\mathcal{L}_2 = \mathbb{E}[D^{(2)}(\tilde{A},\tilde{S})] - \mathbb{E}[D^{(2)}(A,S)] + \lambda_2 \mathbb{E}[(\|\nabla D^{(2)}(\hat{A},\hat{S})\| - 1)^2]$$

where S is the time series of the real trace, $\tilde{S}$ is the time series of the synthetic trace, λ_2 is the gradient penalty coefficient, and $\hat{S}$ is the random interpolation of $\tilde{S}$ and S.

$$\mathcal{L}_3 = \mathbb{E}[D^{(3)}(\tilde{A},wnd(\tilde{S}),\tilde{M})] - \mathbb{E}[D^{(3)}(A,wnd(S),M)] + \lambda_3 \mathbb{E}[(\|\nabla D^{(3)}(\hat{A},wnd(\hat{S}),\hat{M})\| - 1)^2]$$

where M is the record fields of the real trace, $\tilde{M}$ is the record fields of the synthetic trace, λ_3 is the gradient penalty coefficient, $\hat{M}$ is the random interpolation of $\tilde{M}$ and M, and *wnd* is the sliding window function.

D Model and Training Hyperparameters

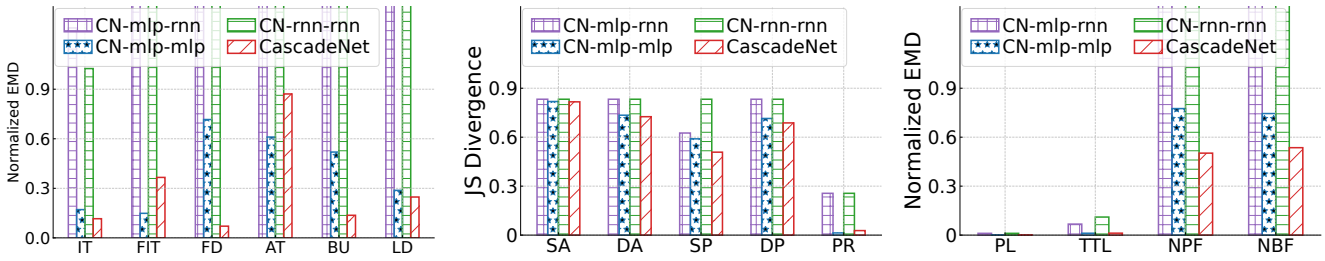
Model Hyperparameters	CAIDA	CA	DC	TON_IoT
Time series length	200	20	100	200
Noise dim	100			
Generator hidden dim	512			
Discriminator hidden dim	512			
Generator MLP layers	3			
Generator LSTM layers	1			
Discriminator layers	5			
Activation function	GELU			
Sliding window	11			

Table 3: Model hyperparameters across datasets.

Training Hyperparameters	CAIDA	CA	DC	TON_IoT
GAN 1 pretraining epochs	200	40	400	4000
GAN 2 pretraining epochs	200	40	400	4000
GAN 3 pretraining epochs	20	20	20	100
Finetuning epochs	20	4	28	240
Batch size	128			
Optimizer	Adam			
Learning rate	1×10^{-4}			
Gradient penalty (WGAN-GP)	10			
Discriminator steps (WGAN-GP)	5			
Coefficient α	1			
Coefficient β	1			

Table 4: Training hyperparameters across datasets.

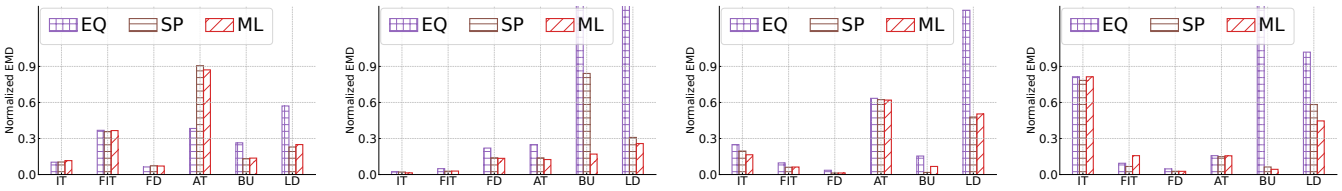
E Comparsion of Different CascadeNet backbone choices



(a) temporal pattern fidelity (b) statistical pattern fidelity

Figure 16: Comparsion of Different CascadeNet backbone choices. The results are based on the CAIDA dataset. **CN-mlp-mlp**: CascadeNet with MLP-based Generator 2 and MLP-based Generator 3; **CN-mlp-rnn**: CascadeNet with MLP-based Generator 2 and RNN-based Generator 3; **CN-rnn-rnn**: CascadeNet with LSTM-based Generator 2 and LSTM-based Generator 3; **CascadeNet** CascadeNet with LSTM-based Generator 2 and MLP-based Generator 3 (standard CascadeNet).

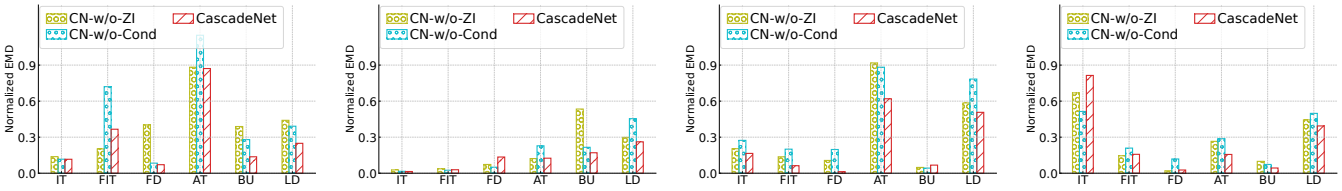
F Impact of Different Timestamp Generation Techniques



(a) CAIDA (b) DC (c) CA (d) TON_IoT

Figure 17: Normalized EMD (↓) between the temporal patterns of real and synthetic traces using different timestamp generation techniques.

G Ablation Study of Optimization Techniques



(a) CAIDA (b) DC (c) CA (d) TON_IoT

Figure 18: Ablation study of optimization techniques. **CN-w/o-ZI**: CascadeNet without zero-inflation method; **CN-w/o-Cond**: CascadeNet without conditional input.

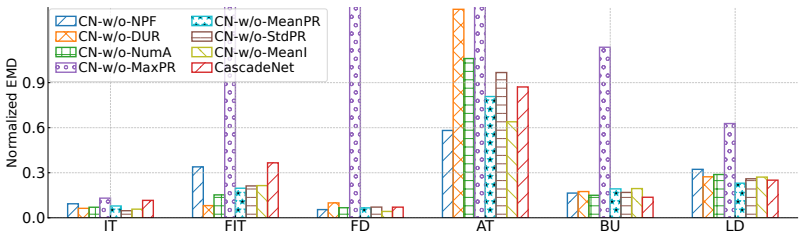
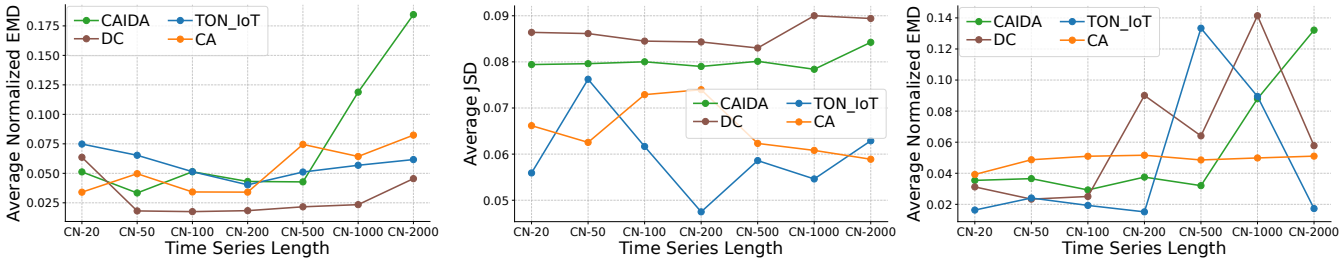


Figure 19: Ablation study of conditional input strategy. The results are based on the CAIDA dataset. **CN-w/o-NPF**: CascadeNet without input feature *number of packets per flow* ; **CN-w/o-DUR**: CascadeNet without input feature *flow duration* ; **CN-w/o-NumA**: CascadeNet without input feature *number of activate time steps* ; **CN-w/o-MaxPR**: CascadeNet without input feature *maximum packet rate* ; **CN-w/o-MeanPR**: CascadeNet without input feature *mean packet rate* ; **CN-w/o-StdPR**: CascadeNet without input feature *standard deviation of packet rate* ; **CN-w/o-MeanI**: CascadeNet without input feature *mean interval between activate time steps* .

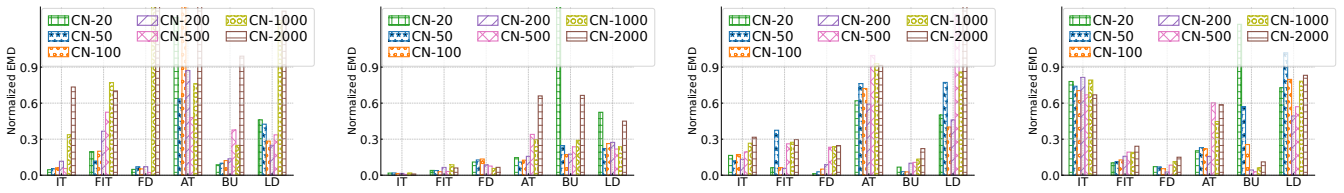
H Impact of Different Time Series Lengths



(a) temporal pattern fidelity

(b) statistical pattern fidelity

Figure 20: Average JSD (↓) and average normalized EMD (↓) between the statistical and temporal patterns of real traces and synthetic traces generated by CascadeNet using different times series length.



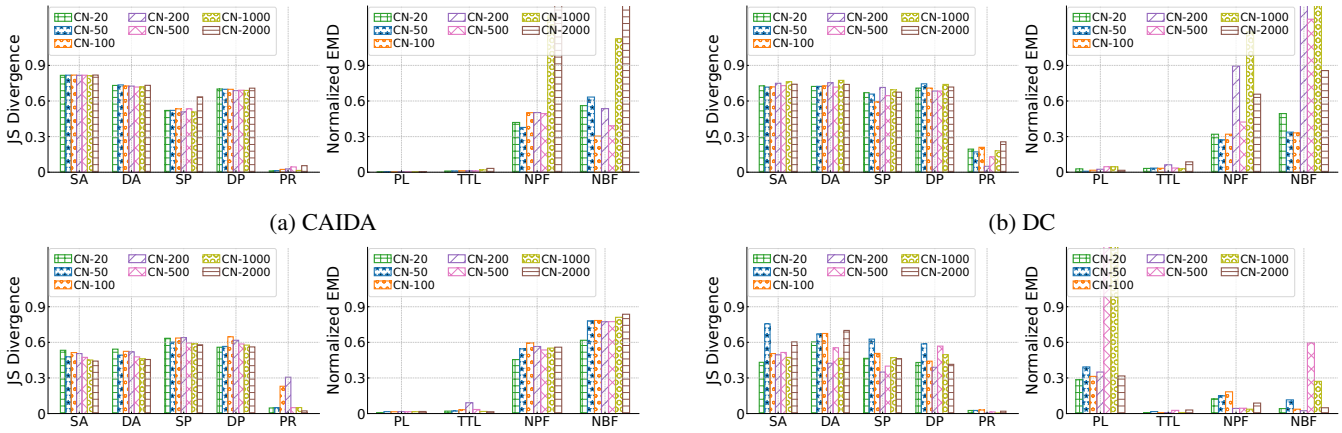
(a) CAIDA

(b) DC

(c) CA

(d) TON_IoT

Figure 21: Normalized EMD (↓) between the temporal patterns of real traces and synthetic traces generated by CascadeNet using different times series length.

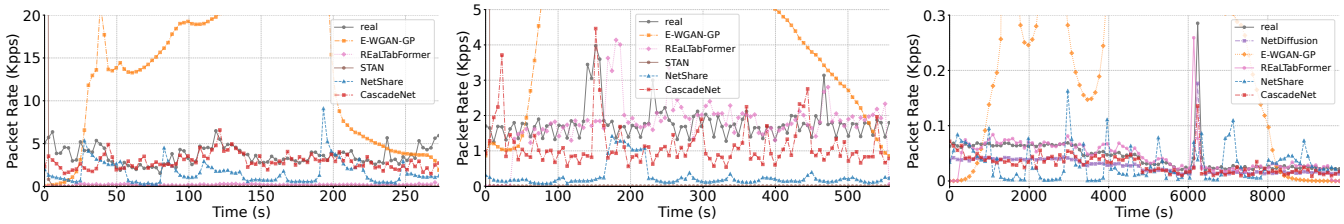


(a) CAIDA (b) DC

(c) CA (d) TON_IoT

Figure 22: JSD (↓) and Normalized EMD (↓) between the statistical patterns of real traces and synthetic traces generated by CascadeNet using different times series length.

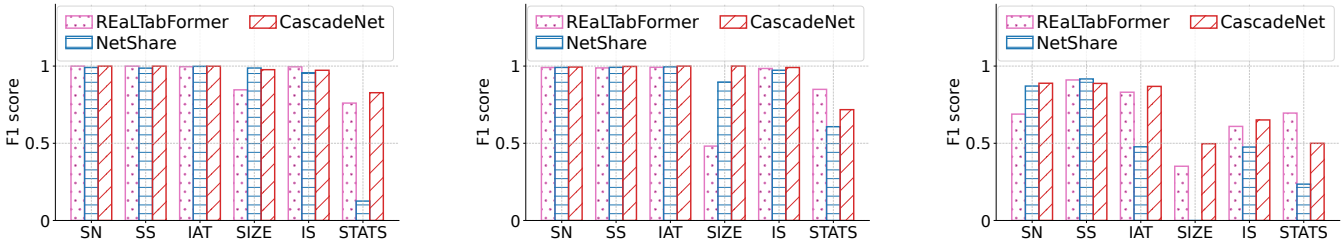
I Packet Rate of Real Traces and Synthetic Traces



(a) DC (b) CA (c) TON_IoT

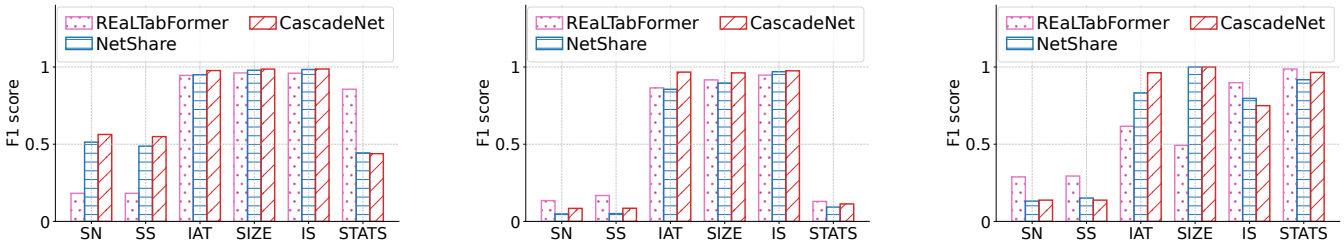
Figure 23: Packet rate of the real traces and synthetic traces generated by different network traffic generators.

J Anomaly Detection by Different Machine Learning Models



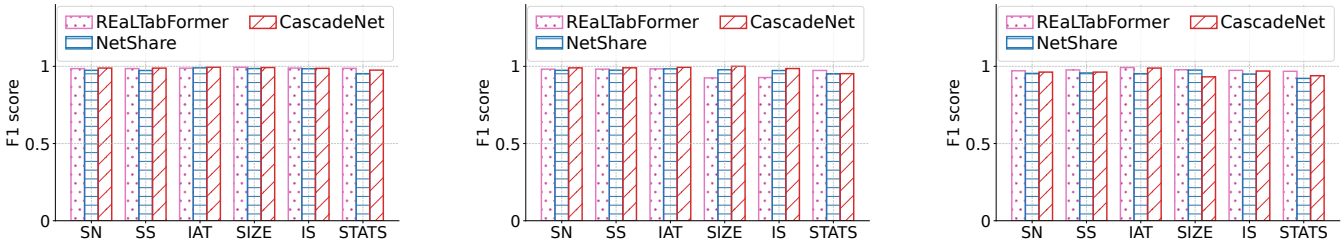
(a) CAIDA (b) DC (c) CA

Figure 24: F1-score (↑) of the prediction by anomaly detection models (AutoEncoder) trained on synthetic traces.



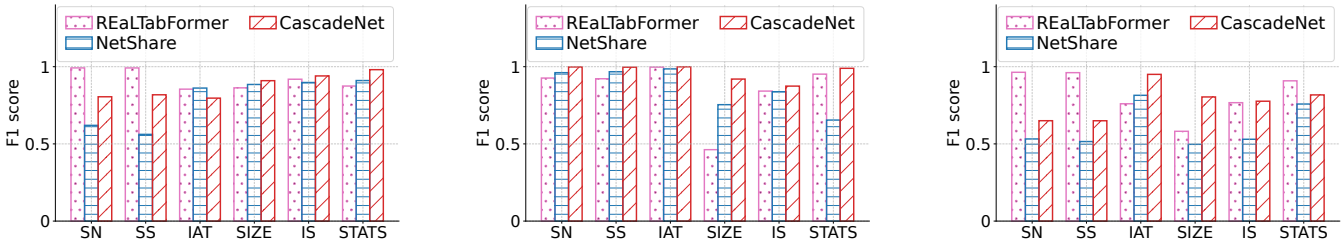
(a) CAIDA (b) DC (c) CA

Figure 25: F1-score (↑) of the prediction by anomaly detection models (GMM) trained on synthetic traces.



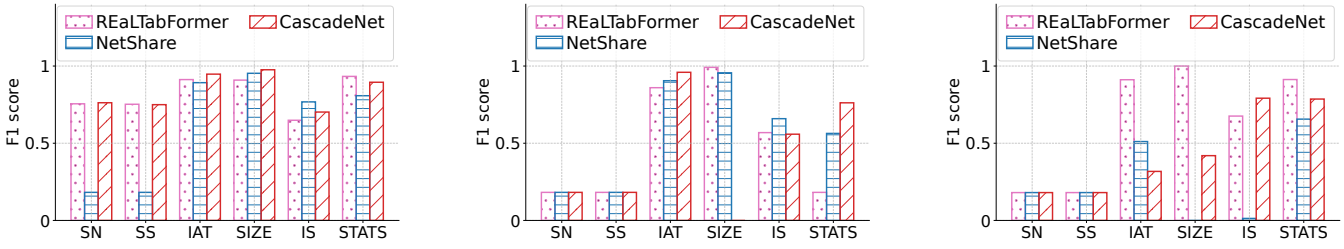
(a) CAIDA (b) DC (c) CA

Figure 26: F1-score (↑) of the prediction by anomaly detection models (IForest) trained on synthetic traces.



(a) CAIDA (b) DC (c) CA

Figure 27: F1-score (↑) of the prediction by anomaly detection models (KDE) trained on synthetic traces.



(a) CAIDA (b) DC (c) CA

Figure 28: F1-score (↑) of the prediction by anomaly detection models (PCA) trained on synthetic traces.