Practical GAN-based Synthetic IP Header Trace Generation using NetShare

Yucheng Yin
Carnegie Mellon University
Pittsburgh, PA
yyin4@andrew.cmu.edu

Zinan Lin
Carnegie Mellon University
Pittsburgh, PA
zinanl@andrew.cmu.edu

Minhao Jin
Carnegie Mellon University
Pittsburgh, PA
minhaoj@andrew.cmu.edu

Giulia Fanti
Carnegie Mellon University
Pittsburgh, PA
gfanti@andrew.cmu.edu

Vyas Sekar
Carnegie Mellon University
Pittsburgh, PA
vsekar@andrew.cmu.edu

ABSTRACT

We explore the feasibility of using Generative Adversarial Networks (GANs) to automatically learn generative models to generate synthetic packet- and flow header traces for networking tasks (e.g., telemetry, anomaly detection, provisioning). We identify key fidelity, scalability, and privacy challenges and tradeoffs in existing GAN-based approaches. By synthesizing domain-specific insights with recent advances in machine learning and privacy, we identify design choices to tackle these challenges. Building on these insights, we develop an end-to-end framework, NetShare. We evaluate NetShare on six diverse packet header traces and find that: (1) across all distributional metrics and traces, it achieves 46% more accuracy than baselines and (2) it meets users' requirements of downstream tasks in evaluating accuracy and rank ordering of candidate approaches.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Security and privacy** → **Data anonymization and sanitization**;

KEYWORDS

synthetic data generation, network packets, network flows, generative adversarial networks, privacy

ACM Reference Format:
Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. 2022. Practical GAN-based Synthetic IP Header Trace Generation using NetShare. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3544216.3544251

1 INTRODUCTION

Packet- and flow-level header traces are critical to many network management workflows. For instance, they are used to guide the design and development of network monitoring algorithms (e.g., [44, 45]), to develop new types of anomaly detection and fingerprinting (e.g., [34, 76, 77]), and to benchmark and test new hardware and software capabilities (e.g., [46]). Unfortunately, access to such traces remains challenging due to business and privacy concerns.

A natural alternative is *synthetic* traces. There is a rich literature in the networking community on generating synthetic traces via simulation-driven approaches (e.g., NS-2 [6]), model-driven approaches (e.g., Harpoon [66] or Swing [70]), and machine learning models (e.g., STAN [75], DoppelGANger [39]), as well as commercial offerings (e.g., IXIA [4]). Unfortunately, existing approaches have notable shortcomings. Model- and simulation-based approaches require significant domain knowledge and human effort to determine critical workload features and configure generation parameters, while not generalizing well across applications [8, 9, 66, 70, 83]. ML-based approaches generalize more easily, but fail to capture domain-specific properties (e.g. packet arrival times, flow length) [39, 75] (§6).

In this work, we explore the feasibility of ML-based synthetic packet-header (e.g., PCAP) and flow-header (e.g., Netflow) trace generation using Generative Adversarial Networks or GANs [12, 28, 29, 39]. If successful, this can lower the barrier for stakeholders with key traces to share synthetic data with potential clients. While the use of GANs is appealing, in practice we find that there are a number of practical challenges in our context that existing approaches (e.g., [21, 31, 39, 57, 71, 74, 75]) fail to satisfy:

- *Fidelity:* Prior techniques (especially those based on tabular data GANs, which dominate the synthetic header generation literature) are unable to capture key correlations across header fields and header fields that have large ranges of values.
- *Scalability-fidelity tradeoff:* Existing techniques require significant GPU-hours to train even moderately-sized traces (e.g., millions of records). Simple tabular GANs take a few hours to train but suffer in fidelity, while more complex time series GANs can take an order of magnitude more time.
- *Privacy-fidelity tradeoffs:* Privacy-fidelity tradeoffs of GANs are not well explored in the context of network header traces. Preliminary work suggests that differentially-private learning approaches are likely to yield poor fidelity for networking datasets [39].

For example, DoppelGANger [39], a state-of-the-art GAN-based approach for time series generation, cannot learn certain key header

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544251

fields (e.g., service ports) well out-of-the-box, while requiring hundreds of GPU hours to train. And while it supports differentially-private (DP) training [25], this option completely destroys its synthetic data fidelity.

In designing NetShare, we tackle these key challenges by a careful data-driven understanding of the limitations of canonical GAN-based approaches. NetShare combines the following key ideas to address the above issues:

- *Reformulation as flow time series generation:* Instead of treating header traces from each measurement epoch as an independent tabular dataset (i.e., rows of packets/flows with headers), we recast the problem for learning synthetic models for a *merged flow-level trace across epochs*. This reformulation allows us to natively capture intra- and inter-epoch correlations.
- *Improving scalability via fine tuning:* We identify opportunities to optimize learning time by using ideas of *model fine tuning* and *data-parallel learning* from the ML literature [81]. Doing so naively may fail to capture dependencies across parallel instances, so we develop heuristics to preserve such correlations.
- *Practical privacy reformulations:* We adopt recent advances in differentially-private model training [82] and combine a small amount of public data with private data to improve privacy-fidelity tradeoffs. To the best of our knowledge, this is the first application and empirical demonstration in the context of header trace generation.

We implement an end-to-end system: NetShare and build a web service prototype available through https://www.pcapshare.com. The code is open-sourced at https://github.com/netsharecmu/NetShare. We also tackle a number of other practical challenges that prior work has not considered. For instance, prior work does not generate valid traces (e.g., headers with derived fields, timestamps), does not evaluate if/how model training generalizes across a wide range of trace sources (e.g., ISP vs. datacenter vs. edge), and does not consider the fidelity of the generated traces for relevant networking use cases (e.g., telemetry, anomaly detection, machine learning).

We empirically evaluate NetShare and show that (1) across all distributional metrics and traces [1, 2, 16, 47, 51, 59], NetShare achieves 46% more accuracy than baseline approaches that use different generative modeling techniques [21, 31, 57, 71, 74, 75]. (2) NetShare meets users' requirements of downstream tasks [19, 22, 44, 45, 77] which keeps the algorithm accuracy and ordering. (3) NetShare achieves a better scalability-fidelity tradeoff than baselines. (4) NetShare can generate higher-quality differentially private traces than baseline approaches.

2 MOTIVATION

In this section, we start by describing use cases for trace-driven analysis in networked systems. Then we argue why synthetic traces are useful and then make a case for data-driven synthesis in contrast to conventional approaches.

2.1 Motivating scenarios

We describe two illustrative use cases in data-driven network design and management that are stymied by lack of access to realistic packet- and flow-level traces.

	Fidelity	Flexibility	Privacy	Effort
Raw	High	✗	✗	Low
Anonymized	Depends	✗	Depends	Low
Synthetic	Possible	High	Possible	High

Table 1: Trade-offs for data holders sharing Raw vs. Anonymized vs. Synthetic traces

Telemetry algorithms. There is a lot of renewed interest in the design and development of novel telemetry algorithms including many approximate data structures for sketching (e.g., [19, 22, 44, 45]). Several of these approaches also make implicit assumptions on structural properties of workloads (e.g., heavy flows) to optimize space-time tradeoffs (e.g., [35, 78]). To systematically evaluate which approach best suits a target deployment or system provisioning regime, we need realistic header traces to compare different algorithms and provisioning strategies (e.g., number of rows, counter arrays to use for sketches).

Evaluating machine learning models:. There are also a number of emerging use cases (including building classifiers over encrypted traffic), where researchers and practitioners (e.g., [34, 63, 76]) are developing novel machine learning models for various types of fingerprinting (e.g., what type of application a particular session entails) or anomaly detection (e.g., is this device compromised) using only IP packet and flow headers [77]. Again, to systematically evaluate the potential performance rate of these algorithms in diverse settings, we need access to realistic header traces of normal client behavior [30, 52].

These use cases (and other future scenarios) require access to *realistic high fidelity* traces. The dimensions of fidelity may be use-case dependent; e.g., some may care about header-field value distributions, some may care about preserving "heavy hitters", others may care about flow-level properties, and so on.

2.2 Synthetic traces and status quo

Due to a number of concerns (e.g., policy, privacy, legal restrictions) data holders who hold traces are usually unwilling to share *raw* traces. To address these concerns, there are two main alternatives to raw packet traces: (1) *Anonymized* traces (e.g., using either masking or cryptographic techniques to hide IP addresses or (2) *Synthetic* traces where some model of generating packet traces is created to mimic properties of the raw data. At a high level, there are qualitative tradeoffs between raw, anonymized, and synthetic trace generation as summarized in Table 1. Raw traces require least effort, but are also least private and least flexible (e.g., generating more data as needed or changing specific workload characteristics). Anonymized and synthetic data each have pros/cons in terms of fidelity, flexibility, privacy, and effort. For example, anonymized data can be made more private by obscuring and/or redacting more fields, but this hurts the resulting data fidelity [50]. Similarly, there are techniques for generating synthetic data, but the resulting privacy guarantees are unclear, and remain an active area of research [18, 32, 41]. Our focus in this paper is on lowering the barrier for generating and sharing synthetic data since it offers a qualitatively different value proposition than the other two options and may lower the barrier for data sharing, as also observed in other efforts (e.g., [39]).

Existing approaches for synthetic header trace generation be divided into three categories: (1) Simulation-based (e.g., NS-2 [6], OSTINATO [7], SEAGULL [9]); (2) Model-driven (e.g., Harpoon [66], SWING [70]); and (3) Data-driven or machine learning driven (e.g., STAN [75]). While these prior efforts have been immensely valuable to the community, they suffer from one or more fundamental shortcomings. The simulation- and model-driven generators have two key drawbacks. First, system designers need to manually determine the important set of features and choose the model which requires significant domain knowledge and human efforts [6, 66, 70]. Second, such models usually make assumptions about the underlying workloads and downstream tasks [66, 70] which makes them hard to generalize across traces with potentially significant deployments/topologies/-workloads. Existing data-driven or machine-learned approaches are more automated but have more fundamental structural limitations. For instance, STAN [75] only generate flow-level summary statistics while HMM-based IP generators [54] only generates IP addresses. Furthermore, existing frameworks do not evaluate the fidelity of these synthetic traces across diverse datasets and downstream tasks.

3 OVERVIEW AND CHALLENGES

Our overarching goal is to develop a data-driven synthetic header trace generation workflow that requires minimal manual tuning and expert knowledge, and can support a wide range of traces from diverse deployments and diverse downstream applications. We start by defining our goals and how we propose to achieve this using a GAN-based workflow.

3.1 Problem formulation

We are given as input a dataset of *header-level traces* split into n consecutive epochs. For each epoch t, we are given $D_t :=$ *unsampled*, IPv4 *packet header trace*. These could be packet- or flow-level traces depending on scenario.

- *Packet header trace:* Each record in a packet header trace consists of packet header fields (e.g., source/destination IP headers) associated with some measured values (e.g., timestamp, packet size).
- *Flow header trace:* Each record in a flow header trace consists of the IP 5-tuple header (e.g., source/destination IP headers, ports, and protocol) associated with some measured values (e.g., start time, end time of flow, total number of packets, total number of bytes).

Scope and goals. Our goal is to learn a *generative model* of $\{D_t : t = 1,2,\cdots,n\}$ that satisfies different types of *fidelity metrics* specified by domain experts and downstream applications. We specifically focus on IPv4 header 5-tuple fields. Packet payloads and other high-layer headers (e.g., TCP/UDP header, application protocol header) are outside of the current scope.

We expect three categories of fidelity metrics of interest:

- **Header-level distributional properties**: For each header field, we want to ensure the distribution of the synthetic and raw trace match quantitatively; e.g., popularity rank of IP addresses or distribution of packet sizes.
- **Flow-level properties**: Other than per-header (or packet-level) metrics, flow-level metrics are also common in networking apps [19, 22, 44, 45, 77]: e.g., flow size distribution or flow duration distribution.
- **Use-case specific properties**: To ensure the utility of synthetic header traces we consider two use-case specific properties: (1) *Accuracy preservation*: Can one particular algorithm/application achieve similar accuracy on the raw and synthetic header traces? (2) *Order preservation*: Is the relative performance of algorithms preserved between raw and synthetic traces; e.g., if Count-Sketch is better for detecting heavy hitters in real traces, is that ordering preserved?

We also want the header traces to satisfy key semantic and syntactic correctness conditions; e.g., IP addresses in valid ranges, packet sizes in ranges (e.g., TCP packet, the minimum size is 40 bytes, while for a UDP packet, the minimum size is 28 bytes); relationship between port number and protocol (e.g., 80 for HTTP and 53 for DNS).

Non goals. We acknowledge some types of properties are out of scope for our current work. Specifically, we do not capture stateful session semantics (e.g., TCP sessions), application layer protocol semantics (e.g., HTTP headers), packet payloads, or fine-grained temporal properties (e.g., distribution of inter-arrival times of packets). These are interesting directions for future work, as we discuss in §8.

3.2 Why GANs

Generative adversarial networks (GANs) are a popular class of generative model [27]. Given a set of training data $x_1,\ldots,x_n$, where samples $x_i \in \mathcal{X}$ belong to universe $\mathcal{X}$ and are drawn from some underlying distribution $x_i \sim P_x$, the goal is to learn to generate new samples from P_x. GANs achieve this through *adversarial training*; that is, they learn two competing models. The *generator* maps low-dimensional random noise to output samples. The *discriminator* takes as input either a real training sample or a generated sample, and must classify which it is seeing. These two models (usually neural networks), are trained in alternation to convergence.

GANs have been used with great success in the image domain, achieving state-of-the-art image and video generation [36, 37]. They are able to learn both local and global correlations in training data to produce high-resolution samples. Hence, there is reason to believe that they may also be good at modeling correlations in network traffic, which involve both short- and long-term correlations [62]. GANs can be tailored to different types of data, including tabular data [74] and time series [26, 39, 80].

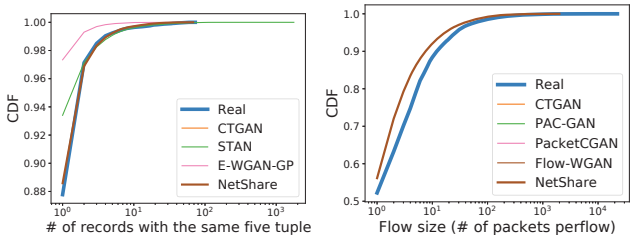
3.3 Strawman approaches and limitations

We begin by understanding the limitations of canonical GAN-based architectures in our context before we explain our design choices to tackle these challenges in the next section.

Strawman solutions. While GANs have most popularly been used for generating image data, they have also been used to generate structured *tabular* data that appear in many application domains [74]. As such, a very natural starting point for using GANs is to treat packet- or flow-header traces as *tabular* data (e.g., CTGAN [74]). Here, each row represents a packet/flow with columns capturing various features of interest (e.g., IP addresses, port, packet/byte counts, timestamp information). Indeed, many existing efforts for extending GAN to networking contexts (e.g., E-WGAN-GP [57]) adopt this

approach with some extensions. A recently proposed GAN architecture called DoppelGANger [39] considers other types of metadata-measurement traces modeled as timeseries. However, it is not clear if, and how, this work can apply to packet- and flow-header traces. As a point of reference, we also consider a state-of-art non-GAN approach called STAN that uses autoregressive neural networks [75]. We defer a full description of these baselines to Section 6.

Challenge 1 (C1): ***Baselines do not accurately capture header correlations of packets/flows, e.g., flow length.***



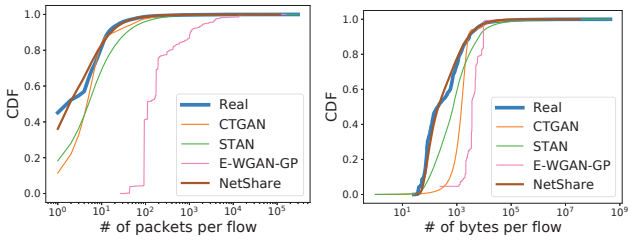
(a) CDF of NetFlow records with same five tuples (UGR16). **(b) CDF of flow size (# of packets) on CAIDA.**

Figure 1: Distribution of # of records/packets with the same five tuples on UGR16 (NetFlow, *left*) and CAIDA (PCAP, *right*). All baselines are missing in Fig. 1b as they don't generate flows with > 1 packet.

Many downstream tasks (e.g., sketch-based telemetry [19, 22, 44, 45], header-based anomaly detection algorithms [77]) need datasets to accurately capture properties that span across packets and flows (e.g., flow size). In the case of packet header traces, we see in Fig. 1b that the baselines are actually absent in the CDF plot of flow size. This is because they do not generate multiple packets for the same flow! This is not surprising as prior GAN-based work has treated each packet as a record in a tabular database, without timestamps [21, 31, 71]. A similar challenge arise with flow data. Long-lived flows can span multiple measurement epochs, and it is not uncommon to see flow records spanning multiple epochs. Moreover, given the way flow collectors are configured (e.g., inactive timeouts, max time of flow), the same flow record can also appear multiple times within a single measurement epoch. As we see in Fig. 1a, baselines either generate much longer flow records (e.g., CTGAN [74], up to a few thousand) or consistently generate short flows (e.g., E-WGAN-GP [57]).

Challenge 2 (C2): ***Baselines struggle to accurately capture the distributions for fields with large support.***

The *support* of a field refers to the possible range of values it can take. Several of the fields we aim to generate have a *large support*, including source/destinations ports, source/destination IPs, and number of packets/bytes per flow. Fields with extremely small/large values could indicate a potential anomaly which are crucial to downstream tasks e.g., anomaly detection [77]. Unfortunately, existing GAN-based baselines do not capture such fields well. Consider the following illustrative examples. In flow-header traces, the "number of packets per flow" and "number of bytes per flow" can range from tens for mice flows to hundreds of millions for elephant flows. Fig.



(a) # of packets per flow **(b) # of bytes per flow**

Figure 2: Distribution of NetFlow's (unbounded) fields on UGR16 dataset: *left*: flow size; *right*: flow volume.

2 shows that baselines generate a much more limited range and also miss the correct distribution for small values. As another example, consider the port number field in headers. Correctly learning the distribution of port numbers (especially the service ports < 1024) is key for many measurement tasks (e.g., anomaly detection [77]). Fig. 3 shows the baselines do not accurately capture the structure of top-K ports (and nearly miss all of them).

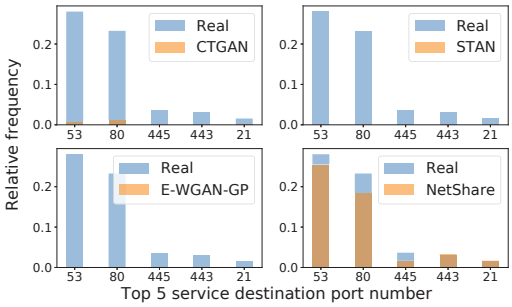
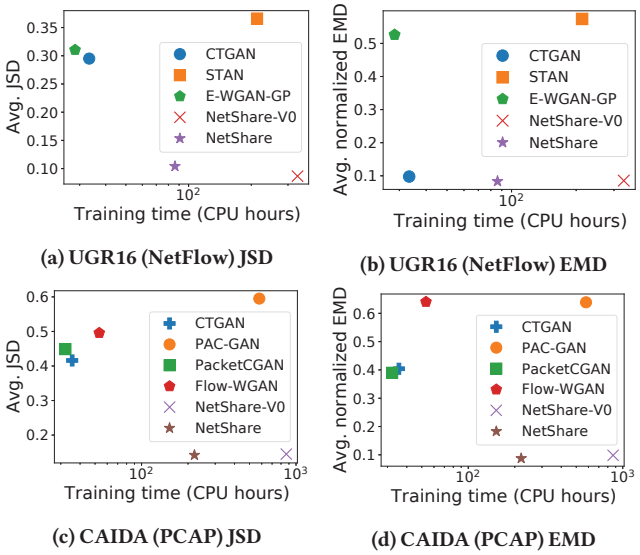


Figure 3: Top 5 service destination ports in TON (NetFlow): baselines fail to capture most frequent service ports while NetShare captures each mode of them by simpler and more effective IP2Vec.

Challenge 3 (C3): ***Existing GAN-based frameworks exhibit poor scalability-fidelity tradeoffs on network traces.***

In theory, some of these fidelity challenges can be partially addressed with larger training datasets, as deep generative models generally achieve better results with more parameters and more data [14]. However, this approach quickly encounters scalability challenges.

Fig. 4 shows the trade-offs between scalability and fidelity of baselines on a NetFlow dataset (Fig. 4a, Fig. 4b) and a PCAP dataset (Fig. 4c, Fig. 4d). We measure scalability as the total CPU hours (as opposed to the wall clock time since multiple machines are used simultaneously) and the fidelity as average JS divergence and normalized EMD across different metrics (refer to Section 6 for details). Simple tabular approaches (e.g., CTGAN, E-WGAN-GP) use the fewest CPU hours while achieving worse fidelity due to their modeling assumptions. We were unable to train the synthetic time series trace generator DoppelGANger [39] on our datasets due to memory constraints. As an intermediate design we modified DoppelGANger to include our proposed merging and encoding techniques (described in §4), shown



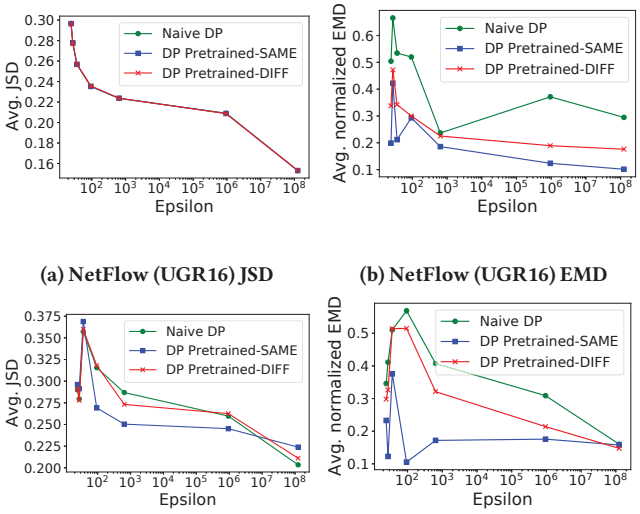
(a) UGR16 (NetFlow) JSD

(b) UGR16 (NetFlow) EMD

(c) CAIDA (PCAP) JSD

(d) CAIDA (PCAP) EMD

Figure 4: Scalability-fidelity trade-offs: Scalability is measured with total CPU hours (↓) and fidelity is measured with the average JSD across categorical fields and the average normalized EMD across continuous fields (↓).



(a) NetFlow (UGR16) JSD

(b) NetFlow (UGR16) EMD

(c) PCAP (CAIDA) JSD

(d) PCAP (CAIDA) EMD

Figure 5: Privacy-fidelity trade-offs: Privacy is measured with (ϵ,δ) in DP (↓) and fidelity is measured as average JSD across categorical fields and the average normalized EMD[1] across continuous fields (↓).

as 'NetShare-V0' in Figure 4. While this can achieve better fidelity, it also uses 10x more CPU hours.

Challenge 4 (C4): *Existing frameworks exhibit poor privacy-fidelity tradeoffs.*

Most prior work on GAN-based trace generation does not evaluate explicit privacy mechanisms [21, 31, 57, 71, 74, 75]. This is inadequate, as synthetic data may present privacy concerns [68]. In the prior work that does explicitly consider privacy [39], the main conclusion is that differentially-private (DP) training via DP-SGD destroys the fidelity of generated signals.[2] Indeed, we can see in Figure 5 that as we decrease the DP privacy parameter ϵ (lower ϵ,δ indicate better privacy; we set $\delta = 10^{-5}$), synthetic data fidelity is destroyed even for weak parameters like $\epsilon = 10^6$ (which means almost no privacy) with an average JS divergence up to 0.21 on UGR16 dataset (Fig. 5a). In other words, even very weak privacy breaks the fidelity. The full experimental setup of Figure 5 is explained in §6.2, Finding 3.

4 NETSHARE DESIGN

Next, we present the design of NetShare via four high-level insights in §4.1 with an end-to-end system overview in §4.2.

4.1 High-level insights

Insight 1 (I1): *We reformulate header trace generation as a time series generation problem of generating flow records for the entire trace rather than a per-epoch tabular approach (Figure 6).*

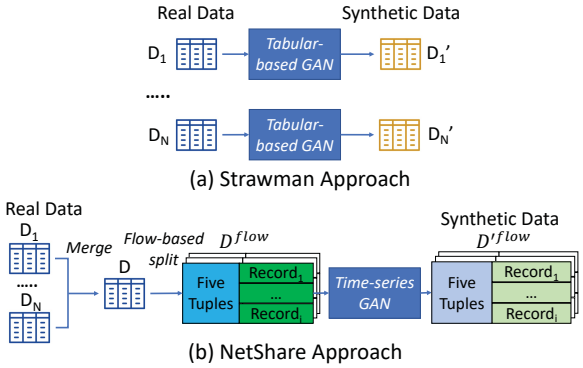


Figure 6: Instead of generating measurement epochs D_i through a tabular GAN, we *merge* multiple epochs D_i into a giant trace D, split the trace into *flows* D^{flow}, and use time-series GAN.

As we saw earlier, existing approaches do not learn header field correlations spanning multiple packets or epochs (e.g., flow size). The root cause is these approaches treat each packet or flow record independently and ignore intra- and inter-*measurement epoch* correlations.

To systematically capture these cross-record correlations, we reformulate the header generation problem as a time series generation problem rather than a tabular generation problem as shown in Figure 6. Specifically, we begin by merging data from measurement epochs D_i into one giant trace D to capture inter-*measurement epoch* correlations. Given this giant trace D, we split it into a set of flows

[1] For each continuous fields, we normalize the EMDs of all models across all epsilons to [0.1,0.9].

[2] We do not argue that DP is necessarily the best or only privacy definition for a networking setting. It is a widely-accepted metric in the privacy community [25]. At the very least, it is natural and desirable to generate DP synthetic data without destroying its fidelity.

fields/embedding	fidelity	scalability	privacy
IP/byte	✗	✓✓	✓
IP/bit	✓	✓	✓
IP/vector	✓✓	✓✓	✗
port/byte	✗	✓✓	✓
port/bit	✓	✓	✓
port/vector	✓✓	✓✓	✓

Table 2: Encoding tradeoffs for various fields. A ✓✓ indicates (qualitatively) good performance on the metric and ✓ indicates acceptable performance. NetShare uses bit encoding for IP and embedded vector representation for port numbers.

D^{flow} based on five-tuples to explicitly capture flow-level metrics (e.g., flow size/duration). Each sample in D^{flow} has a five-tuple as metadata, and a *record* (or "measurement data") corresponding to a sequence of packets for PCAP data and flow records for NetFlow data. Specifically, for PCAP data, each sequence element (packet) includes a *raw* timestamp, packet size, and other IP header fields (we exclude the IP option field and checksum—detailed reasoning in §4.2); for NetFlow, each time series element contains flow start time/duration, packets/bytes per flow, type (attack/benign when applicable).

Finally, we use a time series GAN to model this data. (While autoregressive models [75] also use a time series approach, they are less effective for learning implicit distributions (e.g., flow length [39]), and achieve worse fidelity (§6).) Specifically, we build on an open-source tool called DoppelGANger [39]. Note, however, that natively using a timeseries GAN like DoppelGANger would run into the same issues as the tabular GANs as each flow or packet record will be a timeseries record of length 1 and will miss the key cross-record effects. Furthermore, we will also encounter other challenges regarding encoding, scalability, and privacy.

As shown in Fig. 1, this merge-split-timeseries generation workflow learns a much better flow length distribution compared with baselines. That said, this increases the computational complexity of learning, as seen in Figure 4. We revisit this scalability challenge below.

Insight 2 (I2): ***We use a careful combination of domain knowledge and machine learning to inform the representation of header fields to balance fidelity-privacy-scalability tradeoffs (Table 2).***

Recall that baselines struggle to accurately learn the distribution of fields with large support. Hence, instead of training a GAN on the original data representation, we use domain knowledge to transform certain fields (especially those with large support) into a format that is more tractable for GANs.

For fields with numerical semantics like packets/bytes per flow with a large support, we use log transformation, i.e., $\log(1+x)$ to effectively reduce the range. This simple yet effective technique helps NetShare achieve a better distribution of large-support fields than baselines (Fig. 2). For categorical fields like IP address/port number/protocol, prior work has adopted one or more of the following ideas from the ML literature on feature representation: byte-encoding [21, 31, 71], bit encoding [74], one-hot encoding [75] or advanced vector encoding such as IP2Vec [57] which encodes IPs/-ports/protocols into fixed-length vectors. Unfortunately, these have been adopted without considering robustness across datasets (e.g., number of unique IPs/ports in the dataset), scalability, and privacy. Indeed, while IP2Vec is conceptually appealing (using tools from natural language processing [56, 57]), Fig. 3 shows that E-WGAN-GP, which uses IP2Vec, does not learn the heavy-hitter port distribution.

Table 2 shows a qualitative analysis of different embedding choices for IP/port with respect to fidelity, scalability and privacy. If we look only at fidelity and scalability, a vector embedding of both IP and port (using IP2Vec [56] with careful tuning) outperforms other embeddings. However, if we consider privacy, this approach does not work for a subtle reason. The basic idea of IP2Vec is as follows: as in Word2Vec [49], each five-tuple indexes a *sentence*, and the sequence of IPs, ports, and protocol values are *words*. The collection of five-tuples is used to build a *dictionary* where each unique word (IP or port or protocol) gets mapped to a numeric vector, or *embedding*. The generator is trained on these embeddings; upon generating a new embedding, it is mapped to a word via nearest-neighbor search over the dictionary. However, the dictionary is training data-dependent and therefore not DP.

To resolve this issue, we use *bitwise encodings* of IP addresses while using *IP2Vec* to encode only port numbers and protocols; the embedding was trained on public data (CAIDA backbone trace from a Chicago collector, 2015), which naturally contains almost every possible port number and protocol. In addition, the pairs of ⟨port number, protocol⟩ are diverse enough to cover the common combinations (e.g., 53 for UDP, 80 for TCP). Hence, the IP2Vec mapping is expressive enough to capture the words seen in our private data without violating privacy. As shown in Fig. 3, this variant of IP2Vec captures the top-K service ports (other results are qualitatively similar and not shown for brevity).

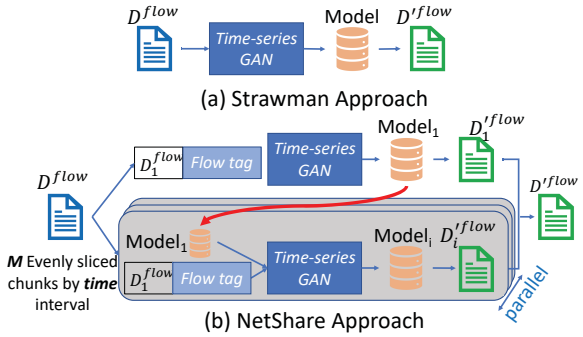


Figure 7: We split D^{flow} into M evenly *time-spaced chunks* with explicit "flow tags" to capture cross-chunk correlations. We use the first chunk as a pre-trained model for parallel training of later chunks.

Insight 3 (I3): ***We can improve the scalability-fidelity tradeoff via fine tuning and parallel training (Fig. 7).***

Recall that reformulating our problem as timeseries generation brings much better header/temporal correlations but increases the total CPU hours. As we can see in Fig. 4, using timeseries GANs to feed the entire giant trace D^{flow} into the generative model increases training time and potentially poses the risk of running out of memory.

One opportunity to improve scalability is via *parallelism*. However, naively dividing the giant trace into *chunks* and parallelizing

training across chunks poses two limitations. First, we again incur the risk of losing correlations across chunks,[3] e.g., flow size distribution for flows that span multiple chunks. Second, while the wall clock time decreases the total CPU hours consumed remains the same.

We avoid these limitations as shown in Fig. 7. First, we borrow the idea of *fine tuning* from the ML literature, i.e., we use a pre-trained model as a "warm start" to seed training for future models [53, 82]. Specifically, we use the first chunk as the "seed" chunk to give a warm-start and subsequent chunks are *fine-tuned* using the model trained from the first chunk. This permits *parallel* training across chunks. One concern remains regarding the cross-chunk correlations; fine tuning alone cannot preserve these. To this end, we append "flow tags" to each flow header to capture the inter-chunk correlation. Specifically, we annotate each flow header with a 0-1 flag denoting whether it starts in "this" chunk. We append a 0-1 vector after the flag with length equal to the total number of chunks, with each bit indicating whether the flow header appears in that specific chunk.

When splitting the giant trace D^{flow} into chunks, we have two natural choices: split by fixed time interval or by number of packets per chunk. Splitting by a fixed number of packets per chunk may impact differential privacy guarantees, as the presence of any single packet could change the final trained model in an unbounded way: removing any packet could change the packet assignment of all following trunks. Thus, we choose to split by fixed time intervals rather than fixed number of packets. We leave the choice of M (number of chunks) as a configurable tradeoff; a higher M would give fewer total CPU hours while increasing the learning complexity across chunks. In our case, we choose $M = 10$ for each dataset with 1 million records.

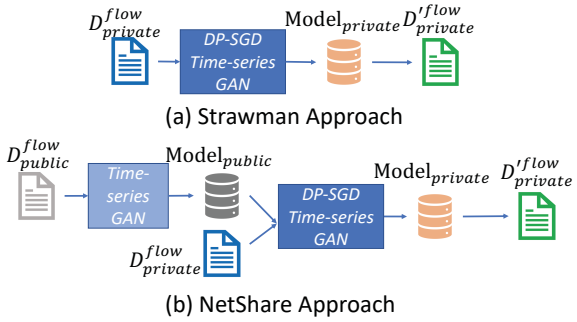


Figure 8: We use public traces to pre-train a public model $Model_{public}$, then fine-tune on private data.

Insight 4 (I4): ***We can improve privacy-fidelity tradeoffs by carefully using public datasets (Fig.8).***

Prior attempts to train DP synthetic network data models using deep generative models have utilized DP-SGD, which modifies stochastic gradient descent (SGD) by clipping each gradient and adding Gaussian noise [11]. For a fixed amount of added noise, the more rounds of DP-SGD we run, the greater the cost in privacy budget. In NetShare, we exploit the observation that one can reduce the number of rounds of DP-SGD needed to achieve a fixed fidelity level by pre-training NetShare on a related public dataset; then, we take the learned parameters from the public dataset, and fine-tune them using DP-SGD over the private dataset. In doing so, we reduce the required number of iterations of DP-SGD. This insight has been explored in related work from the DP community [15, 38, 43, 82], but it has not been utilized in the networking domain to the best of our knowledge. Figure 5 illustrates that this approach can significantly improve the privacy-fidelity tradeoff if used judiciously. We describe the nuances of this approach further in §6.

We also use public data to improve our privacy-fidelity tradeoff due to our IP2Vec encoding. Specifically, we train our IP2Vec mapping on a public dataset with a large number of port/protocol pairs, which helps us learn an embedding without affecting our DP budget (details in §4.1, Insight 2).

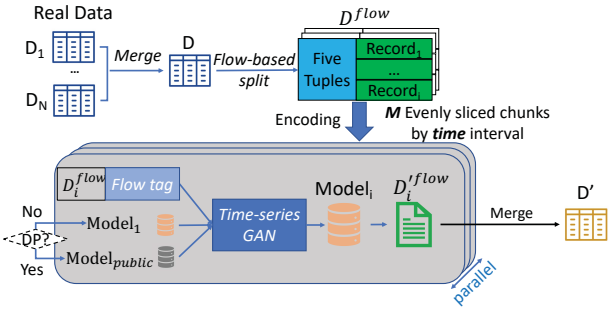


Figure 9: NetShare: end-to-end overview.

4.2 End-to-end view

Combining the key insights above, our end-to-end design is summarized in Fig. 9.

Pre-processing: (Insight 1) Merge data from different *measurement epochs* D_i into one giant trace D with a flow-based split as D^{flow}. (Insight 2): Encode header fields based on domain knowledge and fidelity-scalability-privacy tradeoffs.

Training: (Insights 1/3/4) Evenly slice flow traces into M fixed-*time* chunks with explicit flow tags added. Train a time-series GAN for each chunk; we use DoppelGANger [39] (configuration details in Appendix C). If DP is not required, use the model from the first chunk as the pre-trained model to improve scalability-fidelity tradeoff; If DP is desired, use model pre-trained on public data to fine-tune DP-SGD.

Post-processing: After generating D'^{flow}_i, we map transformed fields back to their natural representations (e.g., map IP2Vec embeddings to ⟨port, protocol⟩ via nearest-neighbor search). Then, we generate derived fields (e.g., checksum).[4] Finally, we convert to PCAP/NetFlow dataset by merging packets/NetFlow records according to the *raw* timestamp (for PCAP) or *raw* flow start time (for NetFlow).

5 IMPLEMENTATION

We implement a prototype of NetShare with Tensorflow 1.15; DP-SGD is implemented using tensorflow-privacy 0.5.0 [10]. In the spirit of reproducible research, we release open source code and detailed documentation at https://github.com/netsharecmu/NetShare.

[3] These chunks are logically independent from the measurement epochs in the original dataset; chunks are merely a construct for parallelizing training.

[4] We make an explicit design choice to exclude such derived fields, which are likely intractable to learn automatically. As such, we use a two-step generation mechanism: (1) use NetShare to generate the native fields (e.g., IP/port/timestamp) and (2) compute the checksum based on that to ensure the correctness of packets. Additionally, we did not take into account the option field in the IP header which is rarely used (and we do not observe the appearance of IP option field in all three PCAP-related datasets).

We also provide a web service prototype available at https://www.pcapshare.com.

For consistent runtime measurement, all experiments are run on the same set of 10 Cloudlab machines [24]. Each machine has Two Intel Xeon Silver 4114 10-core CPUs at 2.20 GHz and 192GB DDR4 memory.

We pre-define a list of relevant hyperparameters (e.g., learning rates, discriminator/generator neural network size, rounds of discriminator/generator alternative training). We evaluated 3-5 options for each hyperparameter, tuned in sequence, prior to running our evaluation. Hyperparameters were tuned over the full training data, as data holders can also do this prior to releasing synthetic data. However, we found that NetShare is not sensitive to these hyperparameters across datasets, and we used the same configuration in all experiments. Our metric for hyperparameter tuning is the relative ordering of Jensen-Shannon Divergence and Earth Mover's Distance between the real and synthetic data for various domain-relevant distributions (details in §6). If downstream tasks are known as a priori, they could be used as one of the "selection criteria" for picking the best model among various hyperparameter setups or training snapshots, which could potentially boost the performance of specific tasks.

We envision data holders sharing the synthetic traces generated from NetShare rather than the learned model [41].[5] We also implement two optional domain-specific privacy extensions that can be applied to the generated traces: (1) IP transformation which transfers synthetic IPs to a user-specified range or a default private range; (2) Specific attributes (e.g., IP addresses/port numbers/protocol) can be retrained to a user-desired distribution to further protect the privacy.

6 EVALUATION

Next, we evaluate NetShare and compare it to existing ML-based synthetic trace generators. We start by describing the datasets and baselines we use.

6.1 Setup

Datasets. In the interest of reproducibility, we select 6 public datasets (3 flow header, 3 packet header). These traces are diverse in the deployments, collection logic, and timescales. For *flow header* datasets, we consider 11 fields in the flow records: (1) source IP address (2) destination IP address (3) source port number (4) destination port number (5) protocol (6) start time of a flow (7) duration of a flow (8) number of packets per flow (9) number of bytes per flow (10) label (if any, benign/attack) (11) attack type (if any, e.g., DoS, brute force, port scans). For *packet header* datasets, we consider the IP header along with the packet arrival timestamp and L4 port numbers (for TCP/UDP only). For each dataset, we evaluate NetShare and baselines on a dataset of 1 million consecutive samples; this is done for consistency with the evaluations in prior work.

- *Flow traces:* **(NetFlow-1) UGR16** [47] consists of traffic (including attacks) from NetFlow v9 collectors in a Spanish ISP network. We used data from the third week of March 2016. The **(NetFlow-2) CIDDS** [58, 59] dataset emulates a small business environment with several clients and servers (e.g., email, web) with injected malicious traffic was executed. Each NetFlow entry recorded with the label (benign/attack) and attack type (DoS, brute force, port scan). The **(NetFlow-3) TON_IoT (TON)** dataset [51] represents telemetry IoT sensors. We use a sub-dataset ("Train_Test_datasets") for evaluating cybersecurity-related ML algorithms; of its 461,013 records, 300,000 (65.07%) are normal, and the rest (34.93%) combine nine evenly-distributed attack types (e.g., backdoor, DDoS, injection, MITM).
- *Packet traces:* The **(PCAP-1) CAIDA:** This dataset [1] contains anonymized traces from high-speed monitors on a commercial backbone link. Our subset is from the New York collector in March 2018. The **(PCAP-2) Data Center (DC)** dataset is a packet capture from the "UNI1" data center studied in the IMC 2010 paper [16]. The **(PCAP-3) Cyber Attack (CA)**: dataset [2] is traces from The U.S. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competitions from March 2012.

Baselines. We compare NetShare to the state-of-the-art GAN-based network traffic synthesizers and a recent auto-regressive-based NetFlow synthesizer [75].[6]

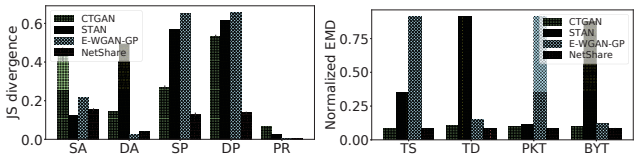
- **CTGAN [74]:** CTGAN is the state-of-the-art GAN for tabular data. While it is not designed for network traffic, we extend it in the following way. We encode IP/port into bits with each bit as a 2-class categorical variable. Other fields are encoded by data type, e.g., timestamp/packet size are treated as continuous fields, protocol is categorical. We use CTGAN as a baseline for NetFlow and PCAP datasets.
- **E-WGAN-GP [57]:** E-WGAN-GP first extends IP2Vec [56] to embed all typical fields in a NetFlow record, i.e., IP address/port/protocol/pkts per flow/bytes per flow/flow start time/flow duration into a fixed-length vector. It then trains a Wasserstein GAN with gradient penalty [29].
- **STAN [75]:** STAN is an autoregressive neural network-based *NetFlow* synthesizer that is designed to capture dependency structures between attributes and across time. STAN groups NetFlow records by host and only ensures correct marginal distributions within the same host. To generate data from multiple hosts, we randomly draw host IPs from the real data.
- **PAC-GAN [21]:** PAC-GAN encodes each network packet into a greyscale image and generates IP packets using Convolutional Neural Network (CNN) GANs. It does not generate packet timestamps and there is no natural way to encode them. Hence, the timestamp is randomly drawn from a Gaussian distribution learned from training data and appended to each synthetic packet.
- **PacketCGAN [71]:** PacketCGAN uses conditional GANs to augment the encrypted traffic datasets which converts each byte of the packet (including the cleartext header) into one bit in the vector. It does not generate timestamps, so we append timestamps to each vector during training.
- **Flow-WGAN [31]:** Flow-WGAN uses Wasserstein GAN [12] on a byte-level embedding. It generates random IP addresses and sets a maximum flow and packet length. Flow-WGAN does not generate timestamps so we again append a timestamp to each byte-embedded vector in training.

[5] Sharing the model reveals more information than a finite number of synthetic data samples. Thus, we posit that in practice stakeholders will be more likely to share synthetic data rather than the models.

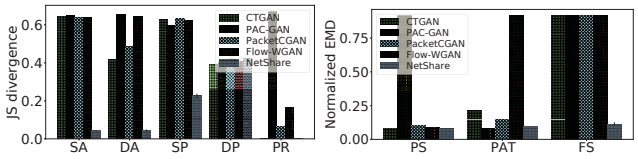
[6] We were unable to reproduce the PcapGAN [23] work as its details and code are lacking.

6.2 Key findings

Finding 1: ***NetShare achieves 46% better fidelity than baselines on feature distribution metrics across traces.***



(a) UGR16 (NetFlow) JSD (b) UGR16 (NetFlow) EMD



(c) CAIDA (PCAP) JSD (d) CAIDA (PCAP) EMD

Figure 10: Jensen-Shannon divergence (↓) and *normalized* Earth Mover's Distance (EMD) (↓) between real and synthetic distributions on UGR16 (NetFlow) and CAIDA (PCAP).

We evaluate the fidelity of synthetic data by computing distance metrics between real and synthetic distributions of various packet- and flow-header fields of interest. The fields include: **SA/DA**: Relative frequency of **S**ource IP/**D**estination IP **A**ddresses ranking from most- to least-frequent; **SP/DP**: **S**ource/**D**estination **P**ort number distribution (from 0 to 65535); **PR**: Relative frequency of IP **Pr**otocol (e.g., TCP/UDP/ICMP). For *NetFlow*-specific metrics, we consider: **TS**: flow start time (in milliseconds); **TD**: flow duration (in milliseconds); **PKT**: number of packets per flow; **BYT**: number of bytes per flow. For *PCAP*-specific metrics, we consider: **PS**: **P**acket **S**ize (in bytes); **PAT**: **P**acket **A**rrival **T**ime (in milliseconds); **FS**: **F**low **S**ize, number of packets per flow. For our distance metrics, we follow common practice in prior work [39, 75]: we use Jensen-Shannon divergence (JSD) for categorical fields (SA/DA, SP/DP, PR), and Earth Mover's Distance (EMD) (also called Wasserstein-1 distance) for continuous fields (TS, TD, PKT, BYT, PS, PAT, FS).[7]. Since EMD has very different scales for different fields, we normalize the EMDs of each field to [0.1,0.9] for better visualization in the figures.

Overall, we find that NetShare is 48% better across NetFlow-related distribution metrics and 41% better across PCAP-related distribution metrics across various traces. Fig. 10 shows a more detailed quantitative comparison of NetShare to baselines on two specific datasets (results on other datasets are qualitatively same, shown in Appendix A). The overall performance of NetShare is consistently better than most baselines. There are cases where NetShare performs worse than some baselines.

[7]Some prior work has also used JSD for continuous fields (e.g., [75]) However, to compute JSD for continuous fields, we need to compute histograms of observed values, and we find that JSD is very sensitive to the bin size. We hence adopt EMD instead, as in [39]. EMD has an intuitive geometric meaning: it is equivalent to the integrated absolute error between the CDFs of the two distributions.

For instance, PAC-GAN appears to achieve perfect packet arrival time distribution across all datasets (e.g., in Fig. 10d). In hindsight, this is not surprising as we explicitly sample packet timestamps from training data *out of band* and append it to the synthetic data.

We also confirmed visually that the structure of these distributions (e.g., CDF and histograms) better match the original raw trace. We do not show these in the interest of brevity and refer readers to the illustrative examples presented in §3.3.

Finding 2: ***NetShare provides better fidelity for downstream network management tasks across different traces.***

We next evaluate whether NetShare synthetic data can be used for downstream applications that utilize different properties of traffic traces. We consider 3 tasks: (1) ML-based *traffic type prediction* on header data; (2) *sketch-based network measurement* [19, 22, 44, 45]; and (3) ML-based *anomaly detection* [77]. For each application, we evaluate: (1) *accuracy preservation* (i.e., if an algorithm performs similarly on real/synthetic data) and (2) *order preservation* (i.e., if algorithms have the same relative performance on real/synthetic data).

Flow-based traffic type prediction. An important use of labeled NetFlow data is to design network traffic type prediction algorithms [51, 58, 59]. We use the fields port number, protocol, bytes/flow, packets/flow, and flow duration to predict the type of a given NetFlow record (e.g., benign/malicious and attack type). We use five common supervised models: Decision Tree (DT), Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB) and Multi-layer Perceptron (MLP). Fig. 11 describes our evaluation setup. We use real data A to generate synthetic data B and B'. Real and synthetic data are sorted by timestamp and split into train:test 80%:20%. Earlier data is used to train the classifier; later data is used for testing.

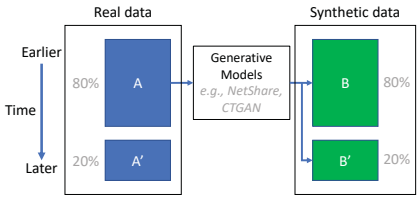


Figure 11: NetFlow traffic type prediction setup

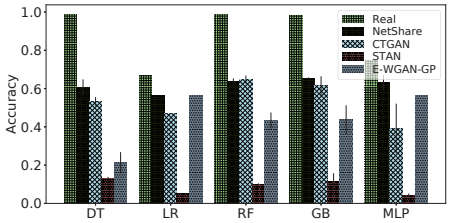


Figure 12: NetFlow traffic type prediction accuracy (↑) on TON: *all* classifiers achieve the highest accuracy with synthetic data generated by NetShare.

We compare the accuracy between training on real (A)/testing on real (A') and training on synthetic (B)/testing on real (A'); this tests the generalization of models trained on synthetic data. Fig. 12

shows results on TON dataset: real data should achieve the highest accuracy. NetShare outperforms all baselines across five classifiers. For example, on the MLP predictor, NetShare achieves 12% higher accuracy than the next-best baselines (E-WGAN-GP) and 84% of the real data accuracy.

Table 3: Rank correlation (↑) of prediction algorithms on CIDDS and TON. Higher is better.

	NetShare	CTGAN	STAN	E-WGAN-GP
CIDDS	**0.90**	0.60	0.60	0.70
TON	**0.70**	0.10	0.60	-0.60

We compare the rankings of models (Decision Tree, Logistic Regression, Random Forest, Gradient Boosting, MLP) when they are trained on real (A)/tested on real (A') vs. trained on synthetic (B)/tested on synthetic (B'). We compute Spearman's rank correlation coefficient between rankings on synthetic and real rankings (1.00 means a perfect match). Table 3 shows that NetShare outperforms all baselines with a higher rank correlation on both CIDDS and TON datasets.

App #2: Sketch-based network telemetry. A growing body of work [19, 22, 44, 45] has studied the use of sketch-based network telemetry.[8] We study a typical downstream task of heavy hitter count estimation, and choose four common sketching algorithms: Count-Min Sketch (CMS) [22], Count Sketch (CS) [22], Universal Monitoring [45], NitroSketch [44]. The threshold for heavy hitters is set at 0.1% with all four sketches use roughly the same memory (our goal is not to compare sketches but to evaluate the value of synthetic traces).

We run the four sketching algorithms on real and synthetic data to get error rates for heavy hitter count estimation $error_{real}$ and $error_{syn}$, which should be equal. We measure their relative error, $\frac{|error_{syn} - error_{real}|}{error_{real}}$, for three heavy hitter counts on three datasets: Destination IP for CAIDA, Source IP for DC, and Five-tuple aggregation for CA. Fig. 13 shows the results on these three datasets. A baseline may be missing for a dataset if the baseline finds no heavy hitters according to the given threshold. For each real/synthetic dataset, every sketching algorithm is independently run 10 times.

NetShare outperforms all valid baselines across different sketching algorithms/heavy hitters of interest/datasets, achieving 48% smaller relative errors on average. We compare the rankings of sketching algorithms' mean heavy hitter estimation error rates, again using Spearman's rank correlation coefficient. NetShare achieves perfect rankings, outperforming the only valid baseline (CTGAN), whose rank correlation can be as low as 0.4 (not shown for brevity).

App #3: Header-based anomaly detection. NetML [77] is a recent open source library for anomaly detection from various flow-based header representations. We use the default one-class support vector machine (OCSVM) and the following supported representations (or "modes") of flows [77]: IAT, SIZE, IAT_SIZE, STATS, SAMP-NUMP (SN), SAMP-SIZE (SS). We defer readers to NetML [77] for a full description.

We run different modes of NetML for real and synthetic data, and get two anomaly ratios: $ratio_{real}$ and $ratio_{syn}$, which should be equal. We compute their relative error, $\frac{|ratio_{syn} - ratio_{real}|}{ratio_{real}}$. For each real/synthetic dataset, every mode of NetML is independently run 5 times. Fig. 14 plots the relative errors for different modes on CAIDA, DC and CA datasets. Note that NetML only processes flows with packet count greater than one, and only baselines that generate such flows are presented in the plots.

Table 4: Rank correlation (↑) of modes of NetML for PCAP anomaly detection.

	NetShare	CTGAN	PAC-GAN	PacketCGAN	Flow-WGAN
CAIDA	**1.00**	N/A	N/A	N/A	N/A
DC	**0.94**	0.43	N/A	N/A	N/A
CA	**0.88**	-0.26	0.37	-0.26	N/A

NetShare outperforms baselines on most datasets and modes of NetML with few exceptions: for mode IAT/SIZE/SS on CA, NetShare achieves the second-best relative error. However, those baselines (e.g., CTGAN, PacketCGAN) are neither robust across datasets, nor do they preserve rankings of NetML modes. Also, their average JSD and normalized EMD across distributional metrics is worse than NetShare (Fig. 10).

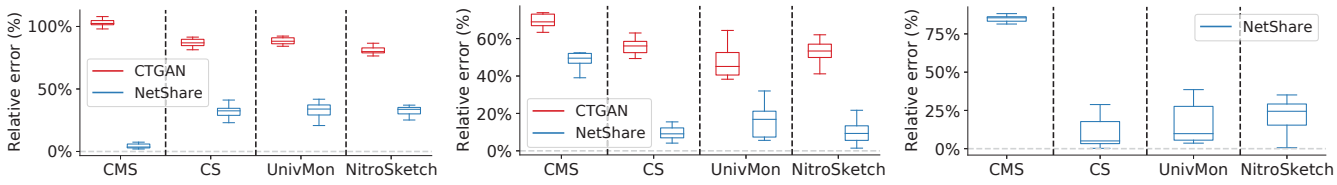
In addition, NetShare outperforms all baselines in terms of preserving the rankings of different modes of NetML. Additionally, compared with groundtruth ranking, NetShare achieves a perfect match on CAIDA. Table 4 shows the exact rank correlations on these three datasets.

Finding 3: ***Pre-training NetShare on public data can improve the fidelity of differentially-private traces.***

As described in §4, we address the challenges associated with training differentially-private (DP) GANs by fine-tuning models trained on public datasets, only using DP optimization (DP-SGD) on the fine-tuning steps. Figure 5 shows that this approach can achieve a better privacy-fidelity tradeoff than the naive approach of training the GAN from scratch with DP-SGD; privacy is measured by DP parameter ϵ (we fix $\delta = 10^{-5}$), and fidelity is measured as the mean JSD across all distributions of categorical fields and the mean normalized EMD across all distributions of continuous fields. This gain is more obvious when the public dataset is similar to the private data; in Figure 5, when the model is pre-trained on a public header trace from a different domain, the privacy-fidelity tradeoff is closer to that of training from scratch. For example, in Figure 5c and Figure 5d, the 'DP Pretrained-SAME' curve was pre-trained on a CAIDA dataset from the Chicago collector in March 2015, and finetuned on our standard "private" CAIDA dataset (New York collector, March 2018). Although these datasets likely see different traffic patterns, they are from the same domain, and we observe significant gains in privacy-fidelity tradeoff. In contrast, the 'DP Pretrained-DIFF' curve was pre-trained on the data center (DC) dataset, and pre-training gives less benefit. This suggests that pre-training can be effective, but care must be taken to select sufficiently close pre-training public datasets.

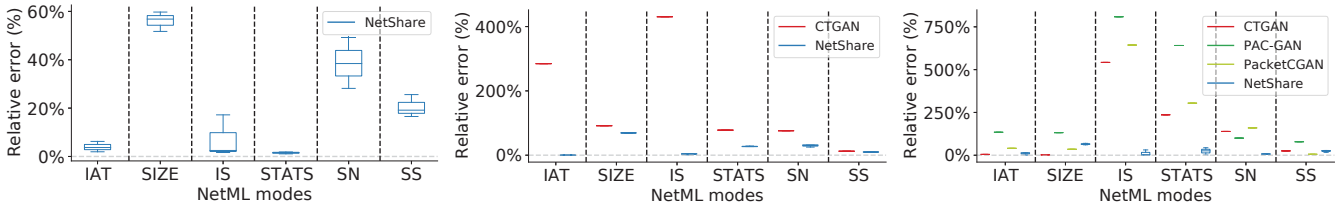
Nonetheless, fine-tuning does not fully resolve the privacy challenges of training DP GAN-based synthetic data. Table 5 shows that for a moderate privacy guarantee ($\epsilon = 24.24$), on the CAIDA dataset, pre-training on a similar public dataset still incurs a 2.3x increase (degradation) in mean EMD (our fidelity metric). This is better than

[8] Sketches uses compact data structures to summarize network traffic.



(a) CAIDA (HH: Destination IP) (b) DC (HH: Source IP) (c) CA (HH: Five-tuple)

Figure 13: Relative error (↓) of heavy hitter count estimation by various sketching algorithms on *real* and *synthetic* PCAP datasets.



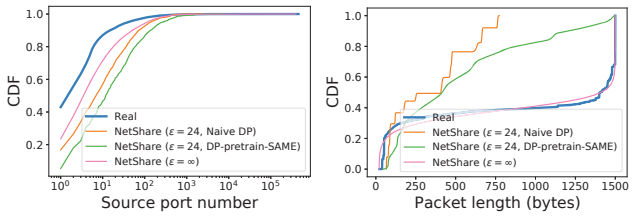
(a) CAIDA (b) DC (c) CA

Figure 14: Relative error (↓) of anomaly detection by NetML modes on *real* and *synthetic* PCAP datasets.

$\epsilon(\delta=10^{-5})$	24.24	27.35	36.28	93.52	641.14	10^6	10^8	w/o DP
Naive DP	0.35	0.41	0.51	0.57	0.41	0.31	0.16	**0.10**
DP-pretrain-SAME	0.23	0.12	0.38	0.11	0.17	0.18	0.16	

Table 5: Normalized EMD (↓) between real and DP synthetic CAIDA data as a function of ϵ (expansion of Fig. 5d).

not pre-training, which incurs a 3.5x increase in mean EMD, but may still be insufficient for practical purposes.



(a) Source port (b) Packet length (bytes)

Figure 15: Packet length and port CDFs computed without noise and under the same (ϵ,δ) with or without pre-training.

We next show how DP affects the distribution of specific packet-level queries in the data. Note that prior work [48] has studied how to build DP analytics pipelines for a pre-specified set of supported queries, including such packet-level queries. However, a fair comparison to [48] is difficult, as NetShare is tackling a harder problem and aims to generate DP synthetic data that can handle *any* type of query.

We conduct two examples of packet-level analysis from [48]: (1) port numbers and (2) packet length. Figure 15a and Figure 15b visualize the distribution of the two fields under different privacy budgets compared with ground truth data. We observe that without adding noise (i.e., $\epsilon = \infty$), NetShare achieves a close match with the real distribution. However, when adding differential privacy (e.g., $\epsilon = 24$), naive DP-SGD training does not give a satisfactory distribution. Though pre-training on datasets from the same domain ("DP-pretrain-SAME") mitigates these problems, it does not resolve the issue. In contrast, [48] reported minimal degradation in query fidelity, even with stronger privacy parameters. Indeed, generating high-dimensional DP synthetic data remains an open question, both in our domain and in general [15, 39, 69, 84].

Finding 4: ***NetShare achieves a better scalability-fidelity tradeoff than baselines.***

Recall from §4 that NetShare trains a model by first splitting the dataset into chunks, then trains a seed GAN model on the first chunk, then fine-tunes that model for successive chunks in parallel. This approach can introduce loss of fidelity, since we are implicitly assuming similarity with the first chunk. However, Figure 4 shows that the resulting scalability-fidelity tradeoff is better than for other baselines. Here, we summarize fidelity by the average JSD across all distributional microbenchmarks on categorical fields and the average normalized EMD across all distributional microbenchmarks on continuous fields, and efficiency by the number of CPU-hours needed to train the model. Particularly for PCAP datasets, we see almost an order of magnitude better JSD compared to the next most-efficient baseline (CTGAN), and almost an order of magnitude faster training compared to the baseline that is closest in fidelity (DoppelGANger).

7 RELATED WORK

Synthetic trace generation has a long history in the networking community. We briefly discuss this related work next.

Network Simulators. Using network simulators to generate traffic traces [6, 7] entails configuring simulators with a number of parameters (e.g., clients, workloads) to ensure these traces match realistic settings. This requires substantial manual effort to extract parameter

from traces, which are often incomplete. Further, most simulators are tied to specific protocols protocols [8, 9, 83] and generalize poorly.

Structural Traffic Generators. A complementary body of trace generation uses structural or statistical models to represent and generate network traffic (e.g., [60, 66, 70, 72]). For example, Harpoon [66] uses a set of distributional parameters extracted from traces to generate flow level traffic that matches both temporal volume characteristics and spatial characteristics (source and destination IP address frequency) of the given trace. Swing [70] extracts key user/session/connection/network level distributions to reproduce the network traffic. LitGen [60] uses a renewal process abstraction to model behavior. Tmix [72] first extracts TCP connection information and creates connection vectors to represent the connection to feed into emulation tools [6]. The key challenge here is to choose an appropriate model and parameters that achieve high fidelity for (possibly unforeseen) downstream tasks. These methods implicitly make assumptions about the workload, which prevents generalization. That said, some generators capture stateful and session-level properties that are currently outside our scope.

Non-GAN ML-based Traffic Generators. STAN [75] uses autoregressive neural models to generate synthetic network traffic in a flow-level while it fails to generate more fine-grained features such as individual packet sizes and arrival times within each flow. Redzovic et. al [54] uses Hidden Markov Models (HMM) to generate only packet sizes and packet interarrival time of IP traffic which are quite limited in the coverage of various packet fields. These are complementary efforts; our work is a systematic application of GANs building on their success in other domains.

GAN-based generators We discussed a number of GAN-based baselines (e.g., [29, 71, 74]) in §6. At a high level, we observe that many of the architectural choices these prior efforts make (e.g., tabular data, using IP2vec, encoding packets as images, ignoring temporal aspects) result in suboptimal fidelity, privacy, and scalability. NetShare uses a timeseries GAN like DoppelGANger [39] as a building block, it does not tackle the specific fidelity, scalability, and privacy challenges that arise in the context f of header traces. GANs can also be used to augment imbalanced datasets in intrusion detection algorithms [64, 73, 79] or for generating malicious/adversarial traffic [20, 42, 55, 76]. While NetShare can be extended to these settings, this is outside our scope.

Other generative models The success of GANs has also inspired a number of other types of generative models for synthetic data generation such as Denoising Diffusion Probabilistic Models [33, 65] and score-based models [67]. In general, these are less mature than GANs for synthetic data generation and their fidelity-privacy-scalability tradeoffs are less well understood. Applying them to the networking domain is an interesting direction for future work.

8 DISCUSSION AND FUTURE WORK

While NetShare lowers the barrier for synthetic header trace generation, it is only a starting point. We conclude with limitations and directions for further research.

Fine-grained temporal properties. While NetShare may potentially capture fine-grained inter-arrival properties, we do not extensively evaluate them or related network management tasks (e.g., congestion control, buffer provisioning) in this paper. We leave this for future work.

Extending NetShare to other protocols. NetShare currently operates over Layer 3 IP headers (plus port numbers) . While we believe NetShare can be extended to support other flow representations (e.g., fbflow [61], AWS VPC flows [5]), supporting higher-layer headers will require supporting stateful protocols (e.g., TCP). The NetShare architecture does not currently support stateful protocols, and is unlikely to naturally learn stateful generation. We hypothesize that supporting stateful protocols will require combining the data-driven generator with domain-specific protocol rules. This is an interesting direction for future work.

Extending NetShare to other downstream tasks. Even constraining to header traces, our scope of downstream tasks is admittedly limited. A natural next step is to evaluate the utility in a broader spectrum of header-based inference tasks; e.g., QoE inference over encrypted headers, device/application fingerprinting from header traces, and so on. Looking forward, we also envision new uses for NetShare including potential avenues for collaborative data augmentation or serving as a toolkit for data holders to contribute to public domain data repositories (e.g., CAIDA [1], CRAWDAD [3]).

Payload data. NetShare does not currently generate payloads, which are much higher-dimensional than the headers generated in this work. We expect that realistic payload generation would be challenging, and require different techniques; e.g., it may be possible to train transformer-based language models [17] over payload data.

Measuring overfitting. In the image domain, overfitted generative models are evaluated by looking for duplicates in the synthetic and training data [13]. In our domain, this metric does not apply: a model may memorize some fields without memorizing others, and it is unclear how to measure packet closeness since fields have different units. Our preliminary analysis by measuring the ratio of overlap between synthetic and real values of src/dst IPs and 5-tuples) suggests that NetShare is not memorizing (not shown), but finding a principled way to measure overfitting in the networking domain is an important question for future work.

Ethical Considerations. We evaluated NetShare on public datasets for reproducibility; this does not raise ethical concerns. In general, systems like NetShare should be used with care to ensure that the privacy requirements of the data holder are accounted for as generative models can memorize and leak individual records [68]. While training a DP-NetShare mitigates this risk, it may not hide other aggregate properties of interest. Thus, actual use of such tools must also take into account data holder's privacy expectations.

ACKNOWLEDGMENTS

We thank our shepherd Hamed Haddadi and the anonymous SIGCOMM reviewers for their insightful feedback on the paper. We thank Haonan Wang for the help with earlier versions of NetShare. This work was supported in part by the National Science Foundation through Convergence Accelerator grant CA-2040675 and RINGS grant 2148359. The authors also acknowledge the generous support of the Sloan Foundation, Intel, Siemens, Bosch, J.P. Morgan Chase, and Cisco.

REFERENCES

[1] [n. d.]. The CAIDA UCSD Anonymized Internet Traces. https://www.caida.org/catalog/datasets/passive_dataset. ([n. d.]). Accessed: 2022-01-30.

[2] [n. d.]. Capture files from Mid-Atlantic CCDC. https://www.netresec.com/?page=MACCDC. ([n. d.]). Accessed: 2022-01-30.

[3] [n. d.]. A Community Resource for Archiving Wireless Data At Dartmouth. https://crawdad.org/. ([n. d.]). Accessed: 2022-06-30.

[4] [n. d.]. IXIA. https://www.keysight.com/us/en/cmp/2020/network-visibility-network-test.html. ([n. d.]). Accessed: 2022-02-02.

[5] [n. d.]. Logging IP traffic with VPC Flow Logs. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html. ([n. d.]). Accessed: 2022-02-02.

[6] [n. d.]. ns-2. http://nsnam.sourceforge.net/wiki/index.php/Main_Page. ([n. d.]). Accessed: 2021-02-10.

[7] [n. d.]. OSTINATO. https://ostinato.org/. ([n. d.]). Accessed: 2021-02-10.

[8] [n. d.]. RUDE&CRUDE. http://rude.sourceforge.net/. ([n. d.]). Accessed: 2021-02-10.

[9] [n. d.]. SEAGULL. http://gull.sourceforge.net/. ([n. d.]). Accessed: 2021-02-10.

[10] [n. d.]. Tensorflow Privacy. https://github.com/tensorflow/privacy. ([n. d.]). Accessed on Feb. 1, 2022.

[11] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[12] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.

[13] Sanjeev Arora and Yi Zhang. 2017. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224* (2017).

[14] Yogesh Balaji, Mohammadmahdi Sajedi, Neha Mukund Kalibhat, Mucong Ding, Dominik Stöger, Mahdi Soltanolkotabi, and Soheil Feizi. 2021. Understanding overparameterization in generative adversarial networks. *arXiv preprint arXiv:2104.05605* (2021).

[15] Raef Bassily, Albert Cheu, Shay Moran, Aleksandar Nikolov, Jonathan Ullman, and Steven Wu. 2020. Private query release assisted by public data. In *International Conference on Machine Learning*. PMLR, 695–703.

[16] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 267–280.

[17] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[18] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*. 267–284.

[19] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 693–703.

[20] Jeremy Charlier, Aman Singh, Gaston Ormazabal, Radu State, and Henning Schulzrinne. 2019. SynGAN: Towards generating synthetic network attacks using GANs. *arXiv preprint arXiv:1908.09899* (2019).

[21] Adriel Cheng. 2019. 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, 0728–0734.

[22] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[23] Baik Dowoo, Yujin Jung, and Changhee Choi. 2019. PcapGAN: Packet capture file generator by style-based generative adversarial networks. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 1149–1154.

[24] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[25] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[26] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633* (2017).

[27] Ian Goodfellow. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[28] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 2672–2680.

[29] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028* (2017).

[30] Steffen Haas, Robin Sommer, and Mathias Fischer. 2020. Zeek-osquery: Host-network correlation for advanced monitoring and intrusion detection. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 248–262.

[31] Luchao Han, Yiqiang Sheng, and Xuewen Zeng. 2019. A packet-length-adjustable attention model based on bytes embedding using flow-wgan for smart cybersecurity. *IEEE Access* 7 (2019), 82913–82926.

[32] J Hayes, L Melis, G Danezis, and E De Cristofaro. 2019. LOGAN: Membership Inference Attacks Against Generative Models. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, Vol. 2019. De Gruyter, 133–152.

[33] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239* (2020).

[34] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2020. New directions in automated traffic analysis. *arXiv preprint arXiv:2008.02695* (2020).

[35] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 113–126.

[36] Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4401–4410.

[37] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8110–8119.

[38] Alexey Kurakin, Steve Chien, Shuang Song, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. 2022. Toward Training at ImageNet Scale with Differential Privacy. (2022). arXiv:cs.LG/2201.12328

[39] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. 2020. Using GANs for Sharing Networked Time Series Data: Challenges, Initial Promise, and Open Questions. In *Proceedings of the ACM Internet Measurement Conference*. 464–483.

[40] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. 2018. Pacgan: The power of two samples in generative adversarial networks. *Advances in neural information processing systems* (2018).

[41] Zinan Lin, Vyas Sekar, and Giulia Fanti. 2021. On the Privacy Properties of GAN-generated Samples. In *AISTATS*.

[42] Zilong Lin, Yong Shi, and Zhi Xue. 2018. Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077* (2018).

[43] Terrance Liu, Giuseppe Vietri, Thomas Steinke, Jonathan Ullman, and Zhiwei Steven Wu. 2021. Leveraging Public Data for Practical Private Query Release. *arXiv preprint arXiv:2102.08598* (2021).

[44] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*. 334–350.

[45] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 101–114.

[46] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.

[47] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. 2018. UGR '16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Computers & Security* 73 (2018), 411–424.

[48] Frank McSherry and Ratul Mahajan. 2010. Differentially-private network trace analysis. *ACM SIGCOMM Computer Communication Review* 40, 4 (2010), 123–134.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[50] Meisam Mohammady, Lingyu Wang, Yuan Hong, Habib Louafi, Makan Pourzandi, and Mourad Debbabi. 2018. Preserving Both Privacy and Utility in Network Trace Anonymization. 459–474. https://doi.org/10.1145/3243734.3243809

[51] Nour Moustafa. 2021. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society* 72 (2021), 102994.

[52] Vern Paxson. 1999. Bro: A system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[53] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[54] Hasan Redžović, Aleksandra Smiljanić, and Milan Bjelica. [n. d.]. IP Traffic Generator Based on Hidden Markov Models. *parameters* 1, 2 ([n. d.]), 1.

[55] Maria Rigaki and Sebastian Garcia. 2018. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 70–75.

[56] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. 2017. Ip2vec: Learning similarities between ip addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 657–666.

[57] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. 2019. Flow-based network traffic generation using generative adversarial networks. *Computers & Security* 82 (2019), 156–172.

[58] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. 2017. Creation of Flow-Based Data Sets for Intrusion Detection. *Journal of Information Warfare* 16 (2017), 40–53. Issue 4.

[59] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. 2017. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*. ACPI, 361–369.

[60] Chloé Rolland, Julien Ridoux, and Bruno Baynat. 2007. LiTGen, a lightweight traffic generator: application to P2P and mail wireless traffic. In *International Conference on Passive and Active Network Measurement*. Springer, 52–62.

[61] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 123–137.

[62] Matthew N. O. Sadiku and Sarhan M. Musa. 2013. *Self-Similarity of Network Traffic*. Springer International Publishing, Cham, 251–265. https://doi.org/10.1007/978-3-319-01646-7_10

[63] Paul Schmitt, Francesco Bronzino, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2020. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. In *ACM SIGMETRICS*, Vol. 3. Boston, Massachusetts, 1–25. https://dl.acm.org/doi/10.1145/3366704?cid=81548029281

[64] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. 2020. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 376–385.

[65] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*. PMLR, 2256–2265.

[66] Joel Sommers, Hyungsuk Kim, and Paul Barford. 2004. Harpoon: A Flow-Level Traffic Generator for Router and Network Tests. *SIGMETRICS Perform. Eval. Rev.* 32, 1 (June 2004), 392. https://doi.org/10.1145/1012888.1005733

[67] Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600* (2019).

[68] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. 2021. Synthetic Data–Anonymisation Groundhog Day. *arXiv preprint arXiv:2011.07018* (2021).

[69] Giuseppe Vietri, Grace Tian, Mark Bun, Thomas Steinke, and Steven Wu. 2020. New oracle-efficient algorithms for private synthetic data release. In *International Conference on Machine Learning*. PMLR, 9765–9774.

[70] Kashi Venkatesh Vishwanath and Amin Vahdat. 2009. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking* 17, 3 (2009), 712–725.

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

[72] Michele C Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F Donelson Smith. 2006. Tmix: a tool for generating realistic TCP application workloads in ns-2. *ACM SIGCOMM Computer Communication Review* 36, 3 (2006), 65–76.

[73] Korakoch Wilailux and Sudsanguan Ngamsuriyaroj. 2021. Novel Bi-directional Flow-based Traffic Generation Framework for IDS Evaluation and Exploratory Data Analysis. *Journal of Information Processing* 29 (2021), 256–265.

[74] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling tabular data using conditional gan. *arXiv preprint arXiv:1907.00503* (2019).

[75] Shengzhe Xu, Manish Marwah, and Naren Ramakrishnan. 2020. STAN: Synthetic Network Traffic Generation using Autoregressive Neural Models. *arXiv preprint arXiv:2009.12740* (2020).

[76] Junhua Yan and Jasleen Kaur. 2018. Feature Selection for Website Fingerprinting. *Proc. Priv. Enhancing Technol.* 2018, 4 (2018), 200–219.

[77] Kun Yang, Samory Kpotufe, and Nick Feamster. 2020. A Comparative Study of Network Traffic Representations for Novelty Detection. *arXiv preprint arXiv:2006.16993* (2020).

[78] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 561–575.

[79] Chuanlong Yin, Yuefei Zhu, Shengli Liu, Jinlong Fei, and Hetong Zhang. 2018. An enhancing framework for botnet detection using generative adversarial networks. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 228–234.

[80] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. 2019. Time-series generative adversarial networks. (2019).

[81] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792* (2014).

[82] Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. 2021. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500* (2021).

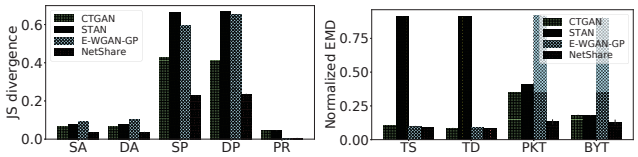
[83] Sebastian Zander, David Kennedy, and Grenville Armitage. 2005. Kute a high performance kernel-based udp traffic engine. (2005).

[84] Zhikun Zhang, Tianhao Wang, Ninghui Li, Jean Honorio, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. 2021. Privsyn: Differentially private data synthesis. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.

APPENDIX

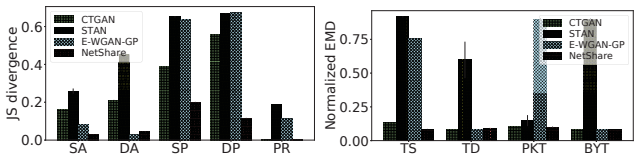
Appendices are supporting material that has not been peer-reviewed.

A ADDITIONAL FIDELITY RESULTS



(a) CIDDS (NetFlow) JSD

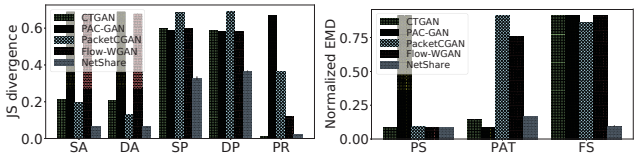
(b) CIDDS (NetFlow) EMD



(c) TON (NetFlow) JSD

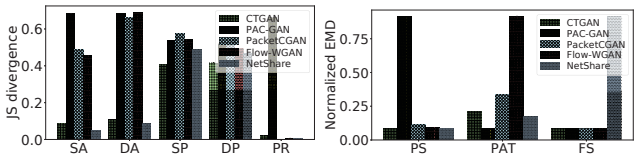
(d) TON (NetFlow) EMD

Figure 16: Jensen-Shannon divergence (↓) and *normalized* Earth Mover's Distance (EMD) (↓) between real and synthetic NetFlow distributions.



(a) DC (PCAP) JSD

(b) DC (PCAP) EMD



(c) CA (PCAP) JSD

(d) CA (PCAP) EMD

Figure 17: Jensen-Shannon divergence (↓) and *normalized* Earth Mover's Distance (EMD) (↓) between real and synthetic PCAP distributions.

Fig. 16 and Fig. 17 show additional results of Jensen-Shannon divergence between real and synthetic datasets that are not shown in §6: NetShare is 48% better across flow-based distribution metrics and 41% better across packet-based distribution metrics across various traces.

B PROTOCOL-COMPLIANT TRACE GENERATION

We also want the packet traces to satisfy key correctness conditions [57, 75] to be valid packet headers. Specifically,

- **Test 1: Validity of IP address.** Source IP address should not be multicast (from 224.0.0.0 to 239.255.255.255) or broadcast (255.xxx.xxx.xxx); Destination IP address should not be of the form 0.xxx.xxx.xxx.
- **Test 2: Relationship between number of bytes (byt) and number of packets (pkt).** (i) For a TCP flow, 40*pkt ≤ byt ≤ 65535*pkt. (ii) Similarly, for a UDP flow, 28*pkt ≤ byt ≤ 65535*pkt.
- **Test 3: Relationship between port number and protocol.** If the port number (e.g., 80 for HTTP and 53 for DNS) indicates one specific type of protocol (TCP/UDP), the protocol field needs to comply with that.
- **Test 4: Packet minimum size (Only valid for PCAP).** For a TCP packet, the minimum size is 40 bytes, while for a UDP packet, the minimum size is 28 bytes.

Table 6: Netflow consistency check on UGR16: NetShare can generate protocol- and domain knowledge-compliant data.

	CTGAN	STAN	E-WGAN-GP	**NetShare**
Test 1	96.90%	90.7%	94.38%	**98.05**%
Test 2	32.38%	93.64%	38.55%	**98.41**%
Test 3	99.37%	**99.91**%	100%	99.90%

Table 7: PCAP consistency check on CAIDA: NetShare can generate protocol and domain knowledge complaint data.

	CTGAN	PAC-GAN	PacketCGAN	Flow-WGAN	**NetShare**
Test 1	**95.59**%	92.98%	92.81%	94.92%	95.06%
Test 2	67.69%	0.02%	**99.16**%	61.50%	76.59%
Test 3	70.18%	0.40%	99.16%	61.50%	**99.77**%
Test 4	99.73%	**99.94**%	99.49%	99.74%	89.71%

Table 6 and Table 7 shows the correctness check results on UGR16 and CAIDA, respectively, with NetShare compared to other baselines. Though NetShare does not achieve the highest correctness on multiple tests, the ratio is still reasonably high. Additionally, baselines that occasionally achieve high correctness do not exhibit good performance in terms of distributional metrics, downstream tasks, scalability-fidelity and privacy-fidelity trade-offs, as we show in §6, which significantly degrades the usefulness of the synthetic datasets generated by baselines.

C IMPLEMENTATION DETAILS OF NETSHARE

For time-series GAN, we use the open source implementation DoppelGANger [39] available at https://github.com/fjxmlzn/doppelGANger with the following configurations:

- Auto-normalization is disabled.
- Auxiliary discriminator is enabled.
- [0,1] normalization for the continuous fields.
- Packing [40] is not used as it empirically does not help improve the fidelity in our context.
- The architecture and the loss function remain the same as DoppelGANger