IoTGemini: Modeling IoT Network Behaviors for Synthetic Traffic Generation

Ruoyu Li, Qing Li, *Senior Member, IEEE*, Qingsong Zou, Dan Zhao, Xiangyi Zeng, Yucheng Huang, Yong Jiang, Feng Lyu, *Senior Member, IEEE*, Gaston Ormazabal, Aman Singh, and Henning Schulzrinne *Fellow, IEEE*

***Abstract*—Synthetic traffic generation can produce sufficient data for model training of various traffic analysis tasks for IoT networks with few costs and ethical concerns. However, with the increasing functionalities of the latest smart devices, existing approaches can neither customize the traffic generation of various device functions nor generate traffic that preserves the sequentiality among packets as the real traffic. To address these limitations, this paper proposes IoTGemini, a novel framework for high-quality IoT traffic generation, which consists of a Device Modeling Module and a Traffic Generation Module. In the Device Modeling Module, we propose a method to obtain the profiles of the device functions and network behaviors, enabling IoTGemini to customize the traffic generation like using a real IoT device. In the Traffic Generation Module, we design a Packet Sequence Generative Adversarial Network (PS-GAN), which can generate synthetic traffic with high fidelity of both per-packet fields and sequential relationships. We set up a real-world IoT testbed to evaluate IoTGemini. The experiment result shows that IoTGemini can achieve great effectiveness in device modeling, high fidelity of synthetic traffic generation, and remarkable usability to downstream tasks on different traffic datasets and downstream traffic analysis tasks.**

***Index Terms*—Internet of Things, synthetic data generation, traffic analysis, generative adversarial networks**

I. Introduction

Nowadays, the world is entering an era of the Internet of Things (IoT). To promote the manageability and security of IoT connections, many machine learning (ML) and deep learning (DL) based *traffic analysis* systems have been proposed, such as monitoring [1], intrusion detection [2]–[4], anomaly detection [5]–[7], and device fingerprinting [8]–[10]. The ML/DL-based methods typically require *adequate* and *diverse* training data for high accuracy. However, access to real-world IoT traffic data can be challenging due to commercial restrictions and privacy concerns raised by IoT device owners, especially given that today's IoT devices collect much sensitive data (e.g., personally identifiable information). While federated learning might mitigate privacy issues (e.g., [11]), it can be challenging to use in many cases due to system heterogeneity and high computational complexity. Workarounds like collecting data from self-built testbeds [12], [13], though practical, can be costly, laborious and hard to scale.

The *synthetic traffic generation* is an alternative solution to the lack of real data in networking community. There have been many simulation tools (e.g., NS-3 [14]) and statistical models (e.g., Swing [15]) that can generate synthetic traffic. However, these tools usually need manual determination of flow attributes and statistics, which relies on expert knowledge and human efforts. In recent years, the state-of-the-art approaches are turning to machine learning models (e.g., STAN [16]) and deep generative models (e.g., PacketCGAN [17]). These approaches can automatically learn the traffic patterns from a batch of real-world traffic, and produce the specified amounts of synthetic traffic with high fidelity. In addition, synthetic traffic also markedly reduces privacy concerns as they do not contain real user information.

Though prior work can achieve good fidelity, we observe two limitations that existing approaches fail to address, which hamper their practical use in the context of IoT traffic:

First, the traffic generation cannot be customized based on certain devices and device functions. With the growing diversity of functions in today's IoT devices, traffic may vary significantly among different devices and even across different functions on the same device. For example, an IP camera mostly generates continuous traffic to a video streaming server for real-time monitoring but produces short burst traffic to an AI cloud for face recognition. Using existing approaches (e.g., [15]–[20]) under this circumstance can inherently generate traffic for dominant functions and ignore traffic for rarely included functions in training data, while some of this traffic can be very useful for certain tasks (e.g., fingerprinting [1]).

Second, existing approaches struggle to generate traffic that combines high fidelity and high usability for downstream tasks. Many approaches can only generate traffic in flow-level records (e.g., [15], [16], [18]–[20]), which cannot be used by a large number of methods for downstream tasks that require packet-level data (e.g., [1], [5], [6], [8], [21]). Though some approaches can generate packet-level traffic (e.g., [17], [22]), they typically generate packets as individuals while cannot preserve sequential relationships. We notice that this information is important for many of the downstream tasks, such as [21] using packet sequences to identify IoT devices.

Ruoyu Li, Qingsong Zou, Yucheng Huang and Yong Jiang are with the Shenzhen International Graduate School, Tsinghua University, Shenzhen, China (e-mail: liry19@mails.tsinghua.edu.cn; zouqs21@mails.tsinghua.edu.cn; huangyc20@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Qing Li and Dan Zhao are with the Department of Strategic and Advanced Interdisciplinary Research, Peng Cheng Laboratory, Shenzhen, China (e-mail: liq@pcl.ac.cn; zhaod01@pcl.ac.cn).

Xiangyi Zeng is with the School of Computer Science and Technology, Xidian University, Xi'an, China (e-mail: xyzeng@stu.xidian.edu.cn).

Feng Lyu is with the School of Computer Science and Engineering, Central South University, Changsha, China (e-mail: fenglyu@csu.edu.cn).

Gaston Ormazabal and Henning Schulzrinne are with the Department of Computer Science, Columbia University, New York, USA (e-mail: gso@cs.columbia.edu; hgs@cs.columbia.edu).

Aman Singh is with Palindrome Technologies, Princeton, USA (e-mail: aman.singh@palindrometech.com).

Corresponding author: Qing Li.

To address these limitations, in this paper, we propose IoTGemini, a novel IoT traffic generation framework that can facilitate the training of downstream traffic analysis models. IoTGemini primarily solves the above two problems through the design of two modules. We first introduce the *Device Modeling Module* that can model all functions and network behaviors of an IoT device purely based upon network traffic. It includes a well-designed heuristic algorithm to profile distinct network behaviors, and the construction of a bipartite graph to depict the mapping relationships between functions and network behaviors. This module enables us to know what traffic the device will generate when a certain function is triggered, laying the foundation for customized traffic generation. We then present the *Traffic Generation Module*, which effectively achieves traffic generation at the packet level. In particular, we design a sequential Generative Adversarial Network called PS-GAN as traffic generators that can generate traffic while preserving the sequential relationships between packets. As PS-GAN generates traffic in raw packet sequence format, it is generic and can be easily transformed into other data formats used by the downstream tasks. By combining the two modules into a pipeline, IoTGemini can work as if we have a real IoT device that a user can freely decide how to use, such as what function is triggered and how long it will be used, and thus the generated traffic can be fully customized.

Comprehensive experiments are conducted to evaluate our framework. We use a real-world IoT testbed and three public datasets, including the traffic of 105 diverse IoT devices. The results show that IoTGemini outperforms existing approaches in terms of multiple metrics for synthetic data generation (e.g., similarity of distribution, sequentiality). Furthermore, we reproduce several recent methods for each of the three typical downstream tasks, including intrusion detection, anomaly detection and device fingerprinting, to demonstrate the feasibility of incorporating IoTGemini into the training of these systems. Our experiments exhibit that using the synthetic traffic generated by IoTGemini can train the models to have a low relative error compared to using the real traffic. In summary, we demonstrate the high fidelity and high usability of the synthetic traffic generated by IoTGemini.

In general, the contributions of this paper are as follows:

- We highlight two key factors of synthetic traffic generation in IoT scenarios, namely customized generation based on distinct usages of device functions, and maintaining packet-level fidelity and usability among traffic.
- We propose a new synthetic traffic generation framework specifically designed for IoT, which consists of two modules to tackle the limitations of existing approaches.
- We conduct a data-driven experiment, reproducing several baseline approaches and multiple recent works for various downstream tasks, showing the high fidelity and high usability of the synthetic traffic generated by IoTGemini.

The rest of the paper is structured as follows. Section II introduces the background and related work; Section III presents the motivations and challenges; Section IV gives our design goals and the overview of IoTGemini; Section V and Section VI elaborate on the Device Modeling Module and the Traffic Generation Module respectively; Section VII discusses the deployment of the framework; Section VIII shows the experiment results; Section IX discusses the limitations and future work; Section X concludes the paper.

II. Background and Related Work

A. ML/DL-based Traffic Analysis and IoT

Recent advance in machine learning and deep learning has revolutionized network traffic analysis tasks, such as intrusion detection [2]–[4], anomaly detection [5]–[7], device fingerprinting [8]–[10], activity identification [21], [23], [24]. One strength of ML/DL-based traffic analysis methods is that, by automatically extracting features from the traffic and using ML/DL models to infer the features, they can greatly reduce the system's reliance on expert knowledge and human labor while achieving high accuracy. A key factor in ensuring their high accuracy is sufficient and diverse training data. For example, in [25] the authors demonstrate that only training on a large number of samples could ensure acceptable performance for many existing works on network intrusion detection.

Besides, different methods may use different features, such as packet-level features, flow-level features and sequence features. For example, Trimananda et al. solely use packet lengths to identify IoT device types and activities [21]. In [2], the authors use flow-level features such as flow volumes and flow duration to monitor IoT systems. Methods like [6], [8] use sequences of multiple packet fields (e.g., sizes, inter-arrival time, protocols) as input and apply deep models (e.g., 1D-CNN, RNN) to perceive the spatial/temporal relationships inside the sequences. Such differences pose challenges to synthetic traffic as a supplement of training data, requiring generated traffic to be as raw (e.g., packet-level fields) and informative (e.g., preserving sequentiality) as possible so as to adapt to different feature extractions.

B. Synthetic Data Generation for Networks

Synthetic data generation is a well-discussed topic in the machine learning and data science community for generating tabular data, images, text, audio, etc. This paper mainly focuses on the synthetic data generation of network traffic. The primitive approach is to use the network simulator tools (e.g., Ostinato [26], RUDE&CRUDE [27], Seagull [28]), which are typically used for functional, load, stress and performance tests by network engineers. There are also IoT simulators to verify the connections between simulated devices and specific cloud services (e.g., Azure [29]). However, these tools require much manual effort to configure the traffic headers and only support a fixed range of protocols, and thus are incapable of automatically modeling real-world traffic patterns.

Based on the format of generated data, approaches for synthetic traffic generation fall into three categories.

Flow-level Data Generation. Many existing works [15], [16], [18]–[20] fall into this category because flow-level data (e.g., 5-tuple, statistics of flow volumes/counts/duration) are common features for many traffic analysis tasks and are more stable to generate. Ring et al. propose a flow-based traffic generation approach by embedding the fields of NetFlow records into a fixed-length vector [20]. Charlier et al. [19] propose to

generate synthetic attack traffic based on the public flow-level attack datasets. However, flow-level features are aggregated from packet-level fields within the same flow, and there is a considerable amount of information loss. Many packet-level features cannot be reversely derived from the generated flow-level data (e.g., length of each packet), reducing the practicality and generality for various downstream tasks.

Packet-level Data Generation. This type of approach generates packet-level representations where each data sample contains the header fields of a packet (e.g., address, port, protocol, size, timestamp, direction). For example, PacketCGAN [17] generates packet-level samples to balance the major and minor encrypted applications in the traffic dataset. In [22], the authors propose PAC-GAN as a packet generator of network traffic using Generative Adversarial Networks. The disadvantage of these approaches is that they generate packets as individual and independent samples, and accordingly cannot express correlations between consecutive packets.

Sequential Data Generation. To deal with the drawback of individual packet data generation, some recent studies [30]–[32] propose to generate sequences of packet-level data representation. For example, in [30], the authors adopt Hidden Markov Models (HMM) to construct an IP traffic generator. Yin et al. use a time-series model to generate time-spaced chunks of traffic for each 5-tuple flow [31]. Though possessing better adaptability to different feature extractions and models of different downstream tasks, this data format is typically more difficult to generate and is an open research field.

From the perspective of methodology, synthetic traffic generation approaches are mainly either based on statistics of distributional parameters (e.g., Harpoon [18], Swing [15]) or ML/DL-based methods (e.g., autoregressive [16], [32], HMM [30]). Among the latter methods, the Generative Adversarial Network (GAN) [33] is a promising approach in recent years especially in the computer vision community. A GAN learns a couple of competing models to improve the quality of generated data in an iterative process.

There are also some works, albeit limited, specifically focused on IoT traffic generation. In [34], the authors propose an IoT traffic generator, which simply outputs a stream of packets of a fixed size, i.e., the average size of the observed packets. Shahid et al. [35] combine an autoencoder with a GAN to produce sequences of packet sizes. However, they do not show the feasibility of their method to generate multivariate packet fields other than packet sizes, and the evaluation is only conducted on a single device. Hui et al. [36] adopt a knowledge-enhanced GAN to generate aggregate statistics of burst traffic data. However, such a coarse-grained format of data cannot adapt to many downstream tasks that require raw packet-level features. Overall, none of these works consider the characteristics of current multifunctional IoT devices and their impacts on traffic generation. Besides, they can only generate a limited number of packet fields or aggregate features, limiting their usability to various downstream tasks.

III. Motivations and Challenges

In this section, we discuss the limitations of prior art for generating IoT traffic and why they are challenging to address.

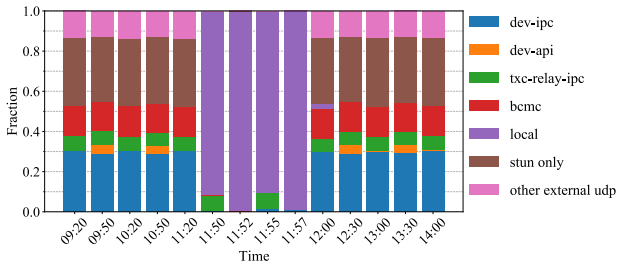


Fig. 1. Traffic distribution of an IP camera communicating with different domains/protocols (common subdomain "tplinkcloud.com" omitted); it maintains idle except between 11:50 and 11:57 that streams to a companion app.

A. Limitations of Prior Art

We highlight two much-desired features for IoT traffic generation that existing work cannot achieve.

Customized Traffic Generation. With various technologies applied to IoT (e.g., AI, edge computing, automation platforms), many IoT devices have evolved from low-functional sensors to integrated multi-functional systems, such as an IP camera with video streaming, face recognition and voice assistant all together. Accordingly, the network traffic to support these functions can be highly different in patterns. Fig. 1 illustrates such an example, which is an IP camera in our testbed. The camera remains idle except between 11:50 and 11:57 when it is awakened by a companion app in LAN for real-time video streaming, which causes a remarkable change in the traffic distribution. For most prior work that simply learns the traffic distribution from a trace dataset, they are inherently biased to generate the most frequent traffic in the dataset, while cannot generate the traffic of arbitrary functions as the customized use of a real IoT device.

Packet-level Usability. Most of the existing packet-level traffic generation approaches can neither generate certain important packet fields nor preserve packet sequential relationships. This limitation hinders their effectiveness for numerous downstream tasks that depend on specific traffic information. For example, some methods for traffic analysis tasks (e.g., [9]) regard destination domains and ports as key information, whereas few prior studies pay attention to the automated generation of these fields. Moreover, some works on identifying IoT activities rely on pattern matching of consecutive packets (e.g., [21], [24]), which require reliable sequentiality between packets. In addition, many recent works on traffic analysis (e.g., [6], [8]) are turning to deep learning models (e.g., CNN, RNN) that are expert in working with spatial/temporal correlated data like images and sequences. To provide data for these models, it is imperative to ensure the sequentiality of the generated data for correct model training.

B. Challenges of Addressing the Limitations

Traffic Heterogeneity vs. Complexity. Prior approaches, whether using statistical models [15], [18] or Conditional GANs [37] that may be suitable for customized data generation, can hardly ensure the generation of traffic heterogeneity, as the models are not generalized enough to produce data with

diverse distributions. This issue can be exacerbated given the huge differences in traffic generated by different functions of an IoT device (as in Fig. 1). A recent approach [31] mitigates this issue by building a generator for each of the 5-tuple flows, which can learn a more concentrated traffic pattern. However, the complexity of this solution can be extremely high considering the large number of flows in some IoT devices (e.g., 2868 flows for Samsung Smart Things in a public dataset [13]).

Deterministic and Random Fields. For one thing, as the communication of IoT devices is typically pre-set by firmware, their destination domains and addresses can be deterministic within a limited number [12], and even small changes in these fields may lead to fundamentally different semantics. For another, fields like port numbers are important to certain tasks but can also change in a pseudorandom manner due to specific port selection mechanisms (e.g., RFC 6056 [38]). For example, the ports used by an IP camera in our testbed for streaming are sequentially 37785, 49839, and 48096. Such random patterns are meaningless and learning them even induces overfitting. Most existing works based on generative models, however, fail to discuss the generation of these packet fields.

Hard to Generate Sequentiality. One important reason for the difficulty of generating sequences of packet-level data is that, different from other data types like images where the values of neighboring pixels typically change gradually, the fields of consecutive packets in a flow can drastically change. For example, a packet with a maximum payload has a packet length of 1500, while the next ACK packet only has a packet length of 40. Most of the models used by prior work, including CNN-GAN [22], WGAN-GP [19], [20] and CGAN [17], may not be suitable for generating such sequential data.

IV. Design Goal and Overview

A. Design Goal, Scope and Assumption

Goals. This paper aims to propose a synthetic traffic generation framework for IoT that tackles the aforementioned limitations and challenges. Specifically, it should achieve the following design goals: 1) Customized traffic generation based on IoT devices and device functions, which can not only generate diverse IoT traffic but also reduce the overall complexity; 2) High fidelity, i.e., synthetic traffic is expected to resemble real traffic by various criteria, especially the sequentiality that most of the existing works do not consider; 3) High usability, i.e., synthetic traffic is generated as sequences of packet-level data that adapt to various downstream traffic analysis systems and achieve the training of high-accuracy models.

Problem Setting. Synthetic traffic generation can serve as a valuable service capable of furnishing substantial traffic data while eliminating privacy concerns of real data, as illustrated in Fig. 2. A service provider is an entity that derives and offers traffic generators based on real-world traffic (i.e., Data-Generator-as-a-Service). It can be an IoT manufacturer or a third-party service provider with the capacity to access physical IoT devices and collect traffic data for the modeling of traffic generators. We assume all IoT traffic collected by the service provider is legitimate, which can be best realized

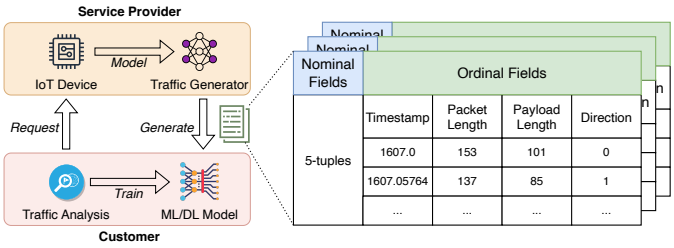


Fig. 2. Workflow of Data-Generator-as-a-Service and target data format.

by locating the IoT devices in a closed environment, such as a LAN with NAT and firewall enabled. A customer aims to develop ML/DL models for specific traffic analysis tasks but lacks necessary access to sufficient physical devices and real traffic data. To augment data resources, customers can request the traffic generators provided by the service provider to generate synthetic data to facilitate model training. To customize the data generation according to their requirements, customers only need to specify parameters such as target devices, utilization patterns, and duration of usage. They do not have to gain an in-depth understanding of the workflows of the traffic generators. Subsequently, customers may perform further feature extraction on the packet-level data produced by the traffic generators to form the ultimate feature set for their tasks (e.g., flow-level features).

Generated Data Format. In alignment with prior packet-level approaches [17], [22], the resultant generated traffic is stored in tabular data (e.g., CSV files) where each row represents an IP packet and each column represents a packet field. The difference lies in our explicit expectation that our generated packets in the same flow to preserve the sequential relationships as the real traffic. As Fig. 2 illustrates, we primarily focus on the generation of two types of packet fields: 1) Ordinal fields, including packet timestamp, packet length, payload length, and directional indicator. For ordinal fields, closer proximity between their values implies a stronger degree of similarity; 2) Nominal fields, which are essentially the 5-tuples of a flow. For nominal fields, closer values do not necessarily suggest a greater degree of similarity. Note that the generation of some fields is not considered in this paper because they are: 1) infrequently used by traffic analysis tasks (e.g., ToS, SEQ number); 2) unavailable by passive sniffing and, as a result, are less commonly utilized, given today's prevalent use of SSL/TLS encrypted traffic (e.g., payload).

B. Overview

The main idea of IoTGemini is that, to achieve customized traffic generation based on device functions, we need to figure out what traffic an IoT device will generate when a certain function is triggered, and find a way to correlate the device functions with a number of network behaviors (e.g., connecting with a certain domain, using a certain protocol). With this knowledge, we can build an individual traffic generator for each network behavior to learn a more concentrated traffic pattern, which can mitigate the challenge brought by traffic heterogeneity within a limited level of complexity. To this

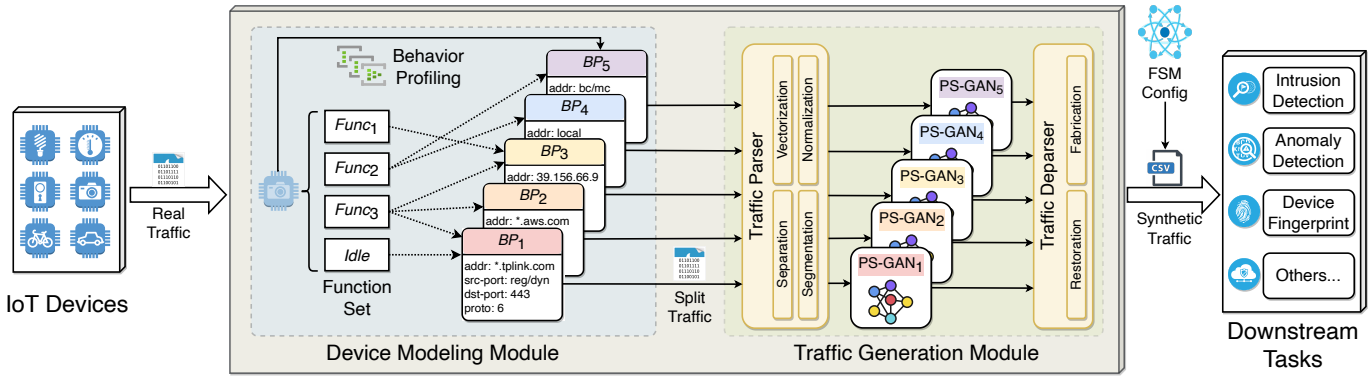


Fig. 3. Overview of IoTGemini.

end, IoTGemini consists of two modules to build an end-to-end pipeline from the modeling of real-world IoT devices and IoT traffic to the generation of synthetic traffic for downstream traffic analysis systems, as illustrated in Fig. 3.

Device Modeling Module. This module is designed to achieve the network-level modeling of IoT devices, specifically, decomposing the functions and network behaviors of a device and determining their correlations. We propose a network behavior profiling method, which combines expert knowledge and traffic statistics to extract a set of Behavior Profiles (BPs) for each network behavior. A BP is similar to 5-tuples in form but different in fact: though a BP also describes the attributes of a device's network connections (e.g., remote domain, port, protocol), it is essentially a coarse-grained abstraction of the flows that are likely to serve the same purpose, which typically has a much smaller quantity than 5-tuples. Moreover, we propose an empirical method to explore the relations between the device functions (e.g., video recording, remote control, movement detection) and the extracted behaviors. Accordingly, the network-level device modeling can be depicted by a bipartite graph that indicates the trigger of a certain function generating traffic under certain BPs, laying the foundation of customized traffic generation.

Traffic Generation Module. Different from existing works that simply learn and generate the overall distribution of a trace dataset, the Traffic Generation Module in IoTGemini builds a generator per BP, which is only made possible by the split traffic from the Device Profiling Module. It gives IoTGemini two advantages: 1) each piece of the split traffic exhibits a more convergent traffic pattern which can mitigate the issue of traffic heterogeneity; 2) customized traffic generation can be realized with a limited number of generators that constrain the overall complexity. We propose a deep learning-based traffic generator, called Packet Sequence GAN (PS-GAN), which is a sequential GAN that can generate sequences of packet-level fields. The generated traffic can keep the original sequential relationships between consecutive packets, improving the fidelity of data distribution. Moreover, the generated traffic has great compatibility with diverse features and models, which enables high usability to downstream tasks.

It is worth noting that the IoTGemini framework introduced above tackles the aforementioned challenges in Section III-B through three key points of design. First, we find that IoT device operations inherently entail transitions between diverse states, each prompted by distinct functionalities. Accordingly, we propose the incorporation of a Finite State Machine (FSM) to encapsulate the customized usage of IoT traffic generators. To the best of our knowledge, this is the first effort to achieve customized IoT traffic generation based on device functions and behaviors. Second, our introduction of Behavior Profiles explicitly delineates whether the 5-tuple fields characterizing network behaviors adhere to deterministic patterns or exhibit randomness within defined ranges. This extra level of granularity facilitates the generation of packet fields with elevated precision. Third, our PS-GAN architecture is crafted to reflect the intrinsic nature of realistic packet-by-packet forwarding with a unidirectional LSTM as the generator, and expert traffic analysis capability with a bidirectional LSTM as the discriminator. This combined approach empowers the generation of packet-level sequentiality with an elevated degree of fidelity.

We elaborate on the two modules in Section V and Section VI respectively, and discuss the deployment of the two modules into an end-to-end pipeline in Section VII.

V. Device Modeling Module

In this section, we introduce the detailed workflow of the Device Modeling Module, which includes function acquisition, behavior profiling and device modeling.

A. Function Acquisition

The latest IoT devices are often integrated with various functions such as remote access, automation platforms, and AI-assisted functions. Since this paper focuses on network-level modeling, we only consider the functions that rely on network connections. We observe the functions of 19 IoT devices in our testbed and categorize them into three types:

- *Transient*: This type of function is for transient purposes, such as retrieving the temperature from a thermometer, or turning on a light by remote control. At the network level, the trigger of such a function will immediately generate a burst of traffic which is typically short flows.
- *Persistent*: This type of function changes the state of a device for a persistent purpose, such as recording videos with a camera, or playing music with a sound box. At

the network level, a long flow is typically used for data transmission between the device and the cloud or the companion app until the function is turned off.
- *Idle*: This is the default state of a device when no function is used. At the network level, a device will still generate certain traffic at the idle state, typically for keepalive connections or default synchronization with the cloud (e.g., a plug reporting the electricity usage).

To comprehensively investigate the network traffic in response to various IoT functions, we consider two approaches for a holistic understanding of different IoT operational characteristics, including *app interactions* and *physical interactions*.

App Interactions. The majority of current IoT devices rely heavily on companion apps. In our testbed consisting of 19 devices, each device is accompanied by such an app. Obtaining device functions that can be initiated through companion apps is straightforward in the case of open-source apps (e.g., openHAB [39]). However, the task becomes more intricate when dealing with proprietary apps, such as Apple Home [40]. In such instances, we lack explicit knowledge regarding which specific interactions initiate network communications. To address this, we employ a fuzzing approach to automate interactions with these apps. It involves the setup of an Android emulator on a personal computer, on which we install the respective IoT apps. Subsequently, we employ an app testing tool, namely Monkey Application Exerciser in conjunction with Android Debug Bridge (ADB) [41], to systematically trigger interactions with all UI components. Concretely, the empirical process includes the following steps:

1) Set up device connections and collect network traffic during its idle state, which typically exhibits sparse activity;

2) Utilize the testing tool to systematically trigger interactions with each UI component of the companion app;

3) For each interaction, we allow a 30-second interval for traffic collection. Then, we terminate the app and iterate this process across all UI components a total of 20 times;

4) Record an interaction as a function if there is a notable surge in device traffic immediately after the interaction;

5) Determine the function type by whether the escalated traffic is a burst or if it persists for the entire interval.

As discussed in prior research [12], identical IoT functions initiated through companion apps can yield disparate network traces, depending on whether the app resides in the same network as the IoT device (i.e., LAN scenarios) or in a distinct network (i.e., remote scenarios). Within our testbed, this phenomenon is observed in 16 out of the 19 devices. Specifically, in LAN scenarios, the companion apps can establish direct communication with the devices utilizing local addresses. Conversely, in remote scenarios, the communication channels necessitate routing through cloud servers. To encapsulate a more comprehensive spectrum of network behaviors, the app interaction experiments on each device are conducted under both scenarios. We regard the functions triggered by each of the scenarios as distinct functions, such as "watch-LAN" and "watch-remote".

TABLE I
PHYSICAL INTERACTIONS WITH IOT DEVICES EQUIPPED WITH SENSORS.

Equipped Sensor	Physical Interaction	IoT Device
camera	movement by humans	webcam
microphone	voice wake-up	speaker, webcam
light sensor	change light	smart bulb
infrared sensor	movement by humans	motion detector (conn. to hub)
temperature sensor	change temperature	thermometer (conn. to hub)
door sensor	open & close door	smart lock (conn. to hub)

Physical Interactions. In addition to app interactions, certain IoT functions are activated through physical interactions which also elicit certain network traffic. These physical interactions mostly revolve around the utilization of specific sensors that perceive changes in surrounding environments. In our study, we primarily consider six commonly deployed sensor types across IoT devices. We design and execute corresponding physical interactions with these devices, as described in Table I. Since these physical interactions cannot be automated through software tools, we manually conduct each interaction five times, with a 30-second interval between iterations for traffic collection. Subsequently, we apply the same logic employed in the app interaction process to discern the functions as well as their respective function types.

B. Behavior Profiling

To further understand the device functions at the network level, behavior profiling is the next step essentially to identify the network flows for the same purpose. Due to the unavailability of the firmware source codes of many IoT devices, even the device owners do not obtain the explicit purpose of each connection (i.e., no ground truth labels). Hence, we reduce the behavior profiling to a traffic clustering problem. Some existing works solve such a problem using traffic statistics [42], [43]. However, their outputs are usually black-box clustering models, which fail to provide interpretable behavior profiles. Besides, common clustering algorithms like k-means need a prior setting of the cluster number, whereas we desire to reduce the cluster number (i.e., behavior number) for a better tradeoff between profiling granularity and complexity.

In IoTGemini, we propose a method that combines expert knowledge and traffic statistics for interpretable behavior profiles. Formally, suppose the full function set of a device generates a series of bidirectional flows $\tau_1, \tau_2, ..., \tau_m$, where τ_i is identified by a 5-tuple $(src, dest, sport, dport, proto)$, our goal is to group the flows into a set of clusters, denoted by B, and minimize the number of clusters:

$$\begin{aligned} &\min\ |\mathrm{B}| \\ &\text{s.t. } C_k(\tau_i, \tau_j) = 1, C_s(\tau_i, \tau_j) = 1, \\ &\qquad \forall \tau_i, \tau_j \in BP, \forall BP \in \mathrm{B}, \end{aligned} \tag{1}$$

where $C_k(\cdot)$ and $C_s(\cdot)$ are the knowledge-based constraint and statistical constraint, respectively.

Condition of knowledge-based constraint. As described in RFC 8520 [44], [45], IoT devices usually connect with a limited set of endpoints using specific protocols for different behaviors. Hence, for two flows τ_i and τ_j, we define

$C_k(\tau_i, \tau_j) = 1$ when the following conditions are all satisfied (otherwise $C_k(\tau_i, \tau_j) = 0$).

- *Domain*: If the destinations of both flows are resolved to domain names, they should have at least the common secondary-level domain (e.g., tplink.com).
- *Address*: If at least one of the destinations is not resolved, they should be in the same address range of "local" (e.g., 192.168.1.0/24), "remote" (i.e., public IP address) or "multicast" (224.0.0.0/4 or addresses ending with 255).
- *Source port*: The source ports of both flows should be equal if they are system ports (i.e., $0 \sim 1023$) or be in the range of registered/dynamic ports (i.e., $1024 \sim 65535$).
- *Destination port*: The destination ports of both flows should be equal if they are system ports, or be in the range of registered ports (i.e., $1024 \sim 32767$), or be in the range of dynamic ports (i.e., $32768 \sim 65535$)[1].
- *L4 Protocol*: The L4 protocols of both flows should be equal (e.g., TCP, UDP).

Condition of statistical constraint. We consider two types of statistics of the flows, i.e., the spatial correlation and the temporal correlation. We define $C_s(\tau_i, \tau_j) = 1$ when the following two conditions are both satisfied (otherwise $C_s(\tau_i, \tau_j) = 0$).

- *Spatial correlation (SC)*: The work in [21] has suggested the packet lengths of IoT connections for one behavior are stably predictable, i.e., flows for the same purpose are spatially correlated with respect to packet lengths. We use the Jaccard similarity coefficient between flows τ_i and τ_j over a threshold d_1 as the criterion, which is defined as

$$SC(\tau_i, \tau_j) = \frac{|P_{\tau_i} \cap P_{\tau_j}|}{|P_{\tau_i} \cup P_{\tau_j}|}, \tag{2}$$

  where P_τ is the packet length set of flow τ and d_1 is empirically set to 0.5.
- *Temporal correlation (TC)*: We use the packet inter-arrival time as the temporal statistics [46]. The two flows should be temporally correlated with respect to the normalized difference of average packet inter-arrival time under a threshold d_2. Formally, the temporal correlation between flow τ_i and τ_j is defined as:

$$TC(\tau_i, \tau_j) = \frac{|T_{\tau_i} - T_{\tau_i}|}{\max(T_{\tau_i}, T_{\tau_j})}, \tag{3}$$

  where T_f is average packet inter-arrival time of the flow τ and d_2 is empirically set to 0.5.

We design a two-step heuristic algorithm to achieve the behavior profiling, as summarized in Algorithm 1. The input of the algorithm is the collected bidirectional 5-tuple set of the device traffic, and the output is the behavior profile set of this device. The first loop achieves the preliminary behavior profiling subject to the knowledge-based constraint (line 3 $\sim$ 8). For each of the 5-tuple, we use an `abstract` function to turn its flow fields to a 4-tuple abstraction of range identifiers (line 4), which can be considered a coarse-grained version of a 5-tuple. The output fields and their optional values include:

[1]The reason to treat destination port differently is that some registered ports can also indicate explicit purposes/services of IoT, such as 1900 for SSDP.

Algorithm 1: Behavior Profiling Algorithm

Input: bidirectional 5-tuple flow set T
Output: behavior profile set B

```
1  // Knowledge-based constraint
2  Initialize an empty hash table H;
3  for τ in T do
4      a ← abstract(τ);
5      if a not in H then
6          H[a] ← list();
7      H[a].append(τ);
8  end for
9  // Statistical constraint
10 Initialize an empty set B;
11 for a in H do
12     for τ_i in H[a] do
13         BP_i ← τ_i[1:] ; // ignore src field
14         for τ_j in H[a] do
15             if C_s(τ_i, τ_j) = 1 then
16                 BP_j ← τ_j[1:];
17                 BP_i ← abstractDiff(BP_i, BP_j);
18                 H[a].remove(τ_j);
19         end for
20         B.add(BP_i);
21     end for
22 end for
23 return B;
```

- `addr`: Derived from *dest* field of the flow; value: {a resolved domain name, "*remote*", "*local*", "*multicast*"}
- `src-port`: Derived from *sport* field of the flow; value: {0∼1023, "*reg/dyn*"}
- `dst-port`: Derived from *dport* field of the flow; value: {0∼1023, "*reg*", "*dyn*"}
- `proto`: Equal to *proto* field of the flow; value: {"*TCP*", "*UDP*"}

Clearly, flows with identical abstraction will be placed together in the same entry of the hash table H, where the key is the 4-tuple and the value is a list to store the 5-tuple flows.

The second loop traverses the hash table to obtain the final behavior profiles subject to the statistical constraint (line 11 $\sim$ 22). We randomly select a flow as an initial template of a BP (line 13), use a greedy strategy to merge other flows under the statistical constraint (line 14 $\sim$ 17) and eventually find a BP that contains as many flows as possible (line 18 $\sim$ 20). The function `abstractDiff` in line 17 merges two BPs by replacing their different fields with range identifiers (e.g., $sport_i = 11125, sport_j = 23266 \rightarrow BP.\texttt{src-port} =$ "reg/dyn"). Particularly, the addresses resolved to domains are merged using the longest common subdomain with a wildcard, such as *.tplink.com. Accordingly, the final number of the clusters (i.e., the number of BPs) can be reduced.

The last step of the Device Modeling Module is to complete the device modeling by linking the device functions with the network behaviors, as we observe that a function can trigger the traffic of multiple behaviors simultaneously. For example, the object detection of a drone not only relies on the

connection to an edge server for high-precision AI inference but also synchronizes the detection result with a cloud server.

Function-behavior correlation. The relationship between functions and behaviors can be described by a bipartite graph $G = (\mathrm{F}, \mathrm{B}, E)$, where F is the set of functions and B is the set of BPs. E denotes the edge between F and B, which is a $(0, 1)$ biadjacency matrix of size $|\mathrm{F}| \times |\mathrm{B}|$. We use the similar approach as in the function acquisition, i.e., we trigger each function for 20 times by companion apps and collect the corresponding traffic. With the obtained set of BPs, each flow of the traffic can be mapped to a BP. For a function $Func_i \in \mathrm{F}$, a behavior BP_j is considered correlated if it occurs in all 20 times of triggers and is thus labeled in E, i.e., $E[i, j] = 1$.

Upon the completion of the device modeling, IoTGemini can simulate the lifecycle of an IoT device's network behaviors and specify the custom use of functions as if operating a physical device. It can be easily realized by a configuration file consisting of the time to trigger a function and the duration, simulating the custom user habit of a device. The triggered functions will further awaken the network behaviors according to the relations in the bipartite graph G.

VI. Traffic Generation Module

This section discusses the Traffic Generation Module of IoTGemini, which includes a traffic parser, generator models (PS-GAN), and a traffic deparser. It targets the generation of synthetic traffic that is on the packet level for the generality of use and sequential in nature to keep the relationships between consecutive packets. We use one piece of the split traffic (i.e., belonging to one BP), denoted by a dataset of raw IP packets D, to elaborate the workflow.

A. Traffic Parser

The traffic parser transforms D to a set of sequential data samples as the training input of the subsequent generator model, which consists of the following steps:

- *Separation*: D is separated into a collection of packet sequences by the bidirectional Netflow [47], which identifies bidirectional 5-tuples as flows. To properly handle both short flows and long flows, we employ the mechanisms of inactive timeout (i.e., no further packets are received in α seconds) and active timeout (i.e., long entries are split for every β seconds) to separate flows into sequences. We set the values of α and β to 30 seconds and 150 seconds, respectively.
- *Vectorization*: For each packet, four ordinal fields are extracted to form a vector $\boldsymbol{p}$: IP length (IL), payload length (PL), direction (DIR, 0 for LAN to WAN and 1 for WAN to LAN) and inter-arrival time (IAT). For the first packet of a sequence, its IAT is set to zero.
- *Segmentation*: Each sequence is segmented or cycle padded to a fixed length of packets as the input of most sequential models requires. A segmented sequence is denoted by $P = [\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_m] \in \mathbb{R}^{m \times 4}$.
- *Normalization*: Dimension $k \in \{1, 2, 3, 4\}$ of the vectors is first standardized to a distribution with zero mean and unit variance by

$$S^k = \frac{P^k - \boldsymbol{\mu}_{\mathrm{D}}^k}{\boldsymbol{\sigma}_{\mathrm{D}}^k}, \tag{4}$$

and then normalized to the range of $[-r, r]$ by

$$\overline{P}^k = 2r \cdot \left(\frac{S^k - \mathbf{min}_{\mathrm{D}}^k}{\mathbf{max}_{\mathrm{D}}^k - \mathbf{min}_{\mathrm{D}}^k} - \frac{1}{2} \right), \tag{5}$$

where $\boldsymbol{\mu}_{\mathrm{D}}$, $\boldsymbol{\sigma}_{\mathrm{D}}$, $\mathbf{min}_{\mathrm{D}}$, $\mathbf{max}_{\mathrm{D}} \in \mathbb{R}^4$ are the mean, standard deviation, minimum and maximum of the dataset D, respectively, and r is a scalar. A vector $\boldsymbol{s}_{\mathrm{D}}$ stores the five statistics for the use of traffic parser and deparser.

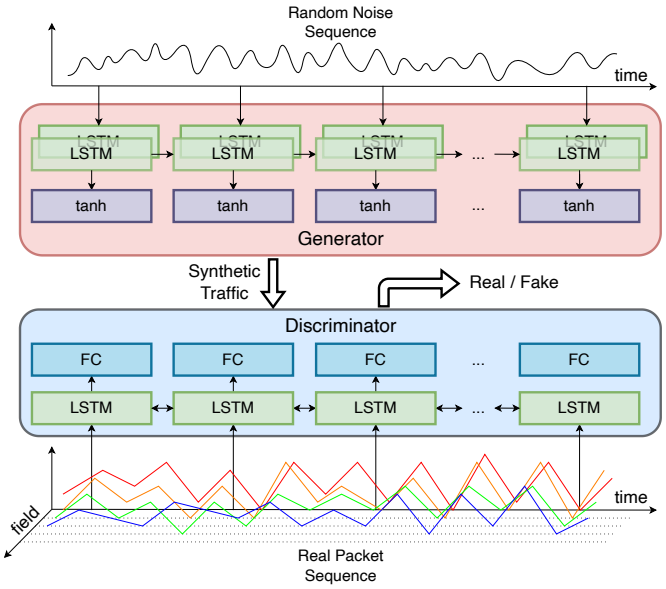


Fig. 4. Architecture of PS-GAN (Packet Sequence GAN).

B. Packet Sequence GAN (PS-GAN)

To generate synthetic traffic of packet-level sequential data, we design PS-GAN, a generative adversarial network (GAN) for sequences of ordinal packet fields. Its architecture is illustrated in Fig. 4. It is essentially a zero-sum game between a generator that attempts to produce packet sequences as real as possible from a noise distribution, and a discriminator as an adversary to distinguish the synthetic data from the real data. With iterations of this competition, the generator learns the mapping from the noise space to the distribution of the real data, and ultimately generates synthetic sequences of good quality. We elaborate on the two components and the process of training and execution as follows.

1) The Generator: We use the multilayer Long Short-Term Memory (LSTM) as the generator. The LSTM [48] is known for its capability of discerning time-series relationships, enabling the generator to synthesize the correlation within consecutive packets (e.g., packets of a TCP bidirectional flow usually alternate between two directions due to ACKs). Formally, for a vectorized packet $\boldsymbol{p}_t$ at position t in a sequence, an

LSTM cell at position t can add or remove information to the cell state by a mechanism called gate, which includes a forget gate (parametrized by W_f), an input gate (parametrized by W_i) and an output gate (parametrized by W_o). By investigating the packet vector $\boldsymbol{p}_t$ and the output of the previous cell h_{t-1}, the workflow of an LSTM cell is as follows.

The forget gate decides the proportion of the old information to be abandoned by

$$f_t = \sigma(W_f \cdot [h_{t-1}, \boldsymbol{p}_t]). \tag{6}$$

The input gate decides the proportion of the new information to be stored by

$$i_t = \sigma(W_i \cdot [h_{t-1}, \boldsymbol{p}_t]). \tag{7}$$

The output gate decides what to output for the next cell by

$$o_t = \sigma(W_o \cdot [h_{t-1}, \boldsymbol{p}_t]). \tag{8}$$

The state of the current cell c_t is calculated by the weighted sum of a candidate state value $\widetilde{c}_t$, calculated as

$$\widetilde{c}_t = \tanh(W_c \cdot [h_{t-1}, \boldsymbol{p}_t]), \tag{9}$$

and the previous cell state

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \widetilde{c}_t. \tag{10}$$

The final output h_t is the multiplication of the output gate and the current cell state with activation and is given as:

$$h_t = o_t \cdot \tanh(c_t). \tag{11}$$

The generator takes a random noise sequence that follows the standard normal distribution as the input and outputs a synthetic sequence by the LSTM and an activation function of hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{12}$$

The output range of the activation function is $(-1, 1)$. The scalar r during the normalization should be less than 1 to make sure the generator can mimic the full values of the real data. In practice, to avoid the vanishing gradient problem happening when the output value is close to the range boundary of activation function $\tanh$, we empirically choose $r = 0.75$.

2) The Discriminator: It incorporates a bidirectional LSTM layer, in conjunction with a fully connected layer. This architecture serves the purpose of evaluating the authenticity of input data by taking into account information from both directions within the sequence. In alignment with the design principles established in the Wasserstein GAN (WGAN) [49], the discriminator in our model does not produce a probability using sigmoid functions but rather outputs a scalar score, aiming to mitigate issues associated with unstable training.

It is worth emphasizing that our selection of distinct model architectures for the generator and the discriminator stems from the intrinsic nature of their respective functions. The unidirectional LSTM employed within the generator is tailored to simulate the packet-by-packet generation process akin to that of a network card. Meanwhile, the bidirectional LSTM utilized in the discriminator is designed to emulate the traffic analysis process executed by an expert, who may consider information from both directions in traffic flows. This nuanced architectural difference aligns with the specific objectives and functionalities of these components within our model.

Algorithm 2: PS-GAN Training Process

Input: Real IP packet dataset D, batch size n
Output: Generator g

1 $\overline{\mathrm{P}}, \boldsymbol{s}_{\mathrm{D}} \leftarrow \mathrm{trafficParser}(\mathrm{D})$;
2 Initialize a generator g and a discriminator d;
3 **while** d *not converged* **do**
4 Sample a batch of real data $\{\overline{P}_i\}_{i=1}^n \subset \overline{\mathrm{P}}$;
5 $\mathcal{L}_d^{real} \leftarrow \frac{1}{n}\sum_{i=1}^n d(\overline{P}_i)$;
6 Sample a batch of noise $\{\boldsymbol{z}_i\}_{i=1}^n \sim \mathcal{N}(0,1)$;
7 $\mathcal{L}_d^{fake} \leftarrow -\frac{1}{n}\sum_{i=1}^n d(g(\boldsymbol{z}_i))$;
8 $\mathcal{L}_d \leftarrow \mathcal{L}_d^{real} + \mathcal{L}_d^{fake}$;
9 $d \leftarrow \mathrm{backUpdate}(d, \mathcal{L}_d^{fake})$;
10 Sample a batch of noise $\{\boldsymbol{z}_i\}_{i=1}^n \sim \mathcal{N}(0,1)$;
11 $\mathcal{L}_g \leftarrow \frac{1}{n}\sum_{i=1}^n d(g(\boldsymbol{z}_i))$;
12 $g \leftarrow \mathrm{backUpdate}(g, \mathcal{L}_g)$;
13 **end while**
14 **return** g;

Algorithm 3: PS-GAN Execution Process

Input: Number of data to generate N, generator g
Output: Synthetic packet dataset $\hat{\mathrm{D}}$

1 Sample N number of noise $\{\boldsymbol{z}_i\}_{i=1}^N \sim \mathcal{N}(0,1)$;
2 $\hat{\mathrm{P}}' = \{\hat{P}'_i\}_{i=1}^N \leftarrow g(\{\boldsymbol{z}_i\}_{i=1}^N)$;
3 $\hat{\mathrm{D}} \leftarrow \mathrm{trafficDeparser}(\hat{\mathrm{P}}', \boldsymbol{s}_{\mathrm{D}})$;
4 **return** $\hat{\mathrm{D}}$;

3) Training: The training process of PS-GAN is described by Algorithm 2. In line 8, the loss function of the discriminator can also be formalized as follows:

$$\mathcal{L}_d = \mathbb{E}_{x\sim p_{real}(x)}[d(x)] + \mathbb{E}_{z\sim\mathcal{N}(0,1)}[-d(g(z))], \tag{13}$$

which means to increase its probability of distinguishing between real and fake data. In line 11, the loss function of the generator can also be formalized as follows:

$$\mathcal{L}_g = \mathbb{E}_{z\sim\mathcal{N}(0,1)}[d(g(z))], \tag{14}$$

which means to increase the probability of fooling the discriminator. Their parameters are updated alternately until reaching a convergence when the discriminator cannot tell between real and fake data[2]. At the end of the training, only the generator is kept for the generation of synthetic traffic during the execution.

4) Execution: The execution process of traffic generation is described by Algorithm 3. During this process, the generator samples the noise data from the standard normal distribution by the given number of synthetic data to generate and outputs the data samples. Each of them is a multivariant sequence that represents the ordinal fields of consecutive packets in a flow.

[2]In practice, the discriminator can be updated multiple times before one update of the generator for better convergence.

In other words, the generator of PS-GAN can generate packet-level data and meanwhile keep the sequential relationships among packets in real traffic, which increases the fidelity of the synthetic traffic and promotes the usability to the downstream systems using various types of features and models.

C. Traffic Deparser

It is responsible for eventually forming the consecutive packet-level data of a flow, which includes the restoration of ordinal fields and the fabrication of nominal fields.

1) Restoration of Ordinal Fields: For the sequences of ordinal fields $\overline{P}' \in \mathbb{R}^{m\times 4}$ generated by the PS-GAN, the restoration process involves the following steps:

- *Denormalization*: It is the inverse process of normalization. For each dimension k of the vectors in $\overline{P}'$, it is realized using the stored vector of statistics $\boldsymbol{s}_{\mathrm{D}}$ by

$$\hat{S}^k = (\mathbf{max}_{\mathrm{D}}^k - \mathbf{min}_{\mathrm{D}}^k)(\frac{\overline{P}'}{2r} + \frac{1}{2}) + \mathbf{min}_{\mathrm{D}}^k, \tag{15}$$

and

$$\hat{P}^k = \hat{S} \cdot \boldsymbol{\sigma}_{\mathrm{D}}^k + \boldsymbol{\mu}_{\mathrm{D}}^k, \tag{16}$$

- *Truncation*: For the fields of IL, PL and DIR, the generated values are truncated to integers as in real packets.

2) Fabrication of Nominal Fields: While PS-GAN effectively generates ordinal fields as part of traffic data, nominal fields are not generated by models due to their inherent determinacy or the presence of randomness that lacks interpretability. As a result, the process of generating nominal fields necessitates a different approach.

We emphasize the notable advantage that our Behavior Profiling confers upon the fabrication of these fields. The 5-tuple fields derived from BPs exhibit a level of precision that sets them apart from the approximately generated values produced by alternative methods [17], [20]. For IP addresses, our approach permits the manual configuration of the device address. The determination of the destination address is facilitated through a DNS query, which resolves the `addr` field within the BP. In cases where a port field is identified as a range identifier within the BP (e.g., "dyn"), it signifies the random utilization of ports to establish distinct flows associated with the same behavior. Consequently, we employ a random value within the specified range as the port number for a given flow sequence (e.g., 32768 to 65535 for "dyn"). Taking into account the specific characteristics of the considered behaviors, this approach ensures that the nominal fields are appropriately fabricated, and contributes to the overall precision of our traffic generation process.

VII. Deployment

A. Usage Configuration

To facilitate the deployment of the two modules within a pipeline for customized traffic generation, we adopt a Finite State Machine (FSM) as a means to configure the manner in which a user interacts with an IoT device. This FSM-driven approach effectively orchestrates the generation of synthetic traffic data in accordance with specified user operations, offering a structured framework for customized traffic generation. An example of presenting such an FSM using a JSON file is as follows:

```
{
 "state0": {
   "dev_ip": "192.168.1.10",
   "start_time": "2023-01-01 12:00:00",
 },
 "states": {
   "func1": {"cond": "12:30", "ac": "5min", "nxt": "idle"},
   "func2": {"cond": "12:50", "ac": "once", "nxt": "idle"},
   "idle": {"cond": "", "ac": "", "nxt": ""}
 },
 "end_time": "2023-01-01 15:00:00"
}
```

Within the provided FSM representation, the initial state, denoted as `state0`, serves the purpose of initializing key parameters such as the device IP address and the start time of the operation. This start time serves as the foundational timestamp for the subsequently generated traffic.

The `states` section comprises various states within the FSM, each of which refers to the utilization of a specific IoT function. Each state is characterized by three key attributes: a condition (`cond`), an action (`ac`), and a next state (`nxt`).

- *Condition* (`cond`): This parameter represents the temporal trigger point for a given function.
- *Action* (`ac`): It signifies the duration of utilization for a persistent function (e.g., "5min") or the frequency of successively triggering a transient function (e.g., "once"). During the execution, our framework employs the bipartite graph in the Device Modeling Module to identify which network behaviors are associated with the triggered function. Subsequently, it activates the generators responsible for these behaviors to produce synthetic traffic data.
- *Next State* (`nxt`): Upon completion of the defined actions, the current state transitions to the next state. The next state then initiates its own traffic generation process.

The traffic generation process concludes when the packet timestamp reaches the specified `end_time`. During the generation of packet data, the absolute timestamp is incrementally calculated using the generated inter-arrival time (IAT) value as an offset, in addition to the initially configured start time.

B. Usage Generation and Validation

To cater to both flexible customized usage and streamlined data generation tests that do not require specific details, we offer three distinct methods for generating configurations in the form of FSMs: 1) manual assignment by users according to their needs and usage patterns; 2) translation from time-based automation rules for users who utilize IoT automation platforms, such as "turn on the light at 7:30 am" of a bulb. Given that these automation rules often share syntactical similarities with FSMs (e.g., trigger, action/event), a parsing script can effectively convert them into our user configuration format; 3) random generation of function triggers, in instances where users seek to introduce variability or simplicity into their data generation tests.

In addition, it is essential to maintain a degree of realism in the generated configurations to ensure the practicality of resulting traffic data. We establish two constraints for the validation of usage configurations: 1) functions triggered by app interactions are constrained to not be activated during late night hours (e.g., between 12 am and 6 am) by default, unless specified otherwise by users; 2) by default, the next state following the execution of a function is set to the idle state, unless manually altered to other states by users. Non-compliance with these constraints may render a user configuration invalid, and the generation process will fail until the user rectifies the configuration. These constraints are in place to ensure that the generated traffic data remains plausible and in accordance with realistic usage patterns.

VIII. Evaluation

In this section, we evaluate IoTGemini by answering the following three questions:

- How is the *effectiveness* of the device modeling, such as the number of functions, the number of BPs and the relations between functions and BPs? (Section VIII-B)
- How is the *fidelity* of our synthetic traffic, i.e., the quality of generated traffic data compared with real IoT traffic data? (Section VIII-C)
- How is the *usability* of our synthetic traffic to various downstream traffic analysis tasks as training data for models? (Section VIII-D)

A. Experimental Setup

IoT Testbed. We build a real-world testbed consisting of 19 IoT devices covering diverse types of devices (e.g., appliance, camera, audio, TV light, IoT hub, sensors) and most mainstream manufacturers in China (e.g., Xiaomi, Huawei, TP-Link). The devices are connected to a wireless router that allows communication with the Internet. All the traffic through the router is collected using a Scapy script, including both the traffic inside the LAN and the traffic between the LAN and the Internet. In addition, as mentioned in Section V-A, a PC is installed with an Android emulator, the IoT apps and an automation script to control the devices for the triggers of their functions and network behaviors. We run the testbed for 10 days and collect a traffic dataset of 10.1 GB in PCAP files.

Datasets. Four real-world IoT traffic datasets are used for the evaluation of traffic generation as Table II describes; the complete device list is in our appendix. In addition to our own dataset, we include three public IoT traffic datasets as benchmarks. As collected in different regions (United States, United Kingdom, and Australia), using different device types (e.g., smart TV, smart brewer) and manufacturers (e.g., Amazon, Google, Apple, Samsung) from ours, they improve the generality of the experiment. We denote the datasets by the abbreviation of the regions in the rest of the paper. All of these datasets are in PCAP files while the public datasets truncate the packet payload. Each of the datasets is randomly split by a ratio of 6:4 for training and testing. The training dataset is used for the training process of data generators, and the testing dataset is used as the real data for the comparison of fidelity to the synthetic traffic and the testing of downstream systems.

TABLE II
Description of IoT Traffic Datasets.

Dataset Notation	PCAP Size	Time Span	Device Number	Packet Number	Flow Number
CN (ours)	10.1 GB	10 days	19	5,849,411	344,983
US [12]	1.8 GB	3 days	41	1,872,002	186,339
UK [12]	488 MB	3 days	33	1,588,821	203,693
AU [13]	2.7 GB	10 days	12	9,615,898	204,181

Besides the IoT traffic datasets, during the experiment on downstream tasks like intrusion detection or anomaly detection, two attack datasets are used as the samples to be detected. One dataset provides the traffic of IoT malware including Mirai and Bashlite [50], and the other dataset provides the traffic of three types of DDoS attacks, two types of scanning attacks and two types of data exfiltration attacks [51]. Both datasets are also stored in PCAP files.

Baselines. To highlight the improvement of IoTGemini, we compare the synthetic traffic of IoTGemini to the synthetic traffic of three types of prior works as baselines, including two approaches for packet-level data (i.e., independent packets in tabular data) generation and one approach for sequential data generation:

- *Gaussian Copula (GC)* [52]: A statistical approach for synthetic data generation. It uses copula functions to model the cumulative multivariate distribution of each field.
- *Conditional Tabular GAN (CTGAN)* [37]: A state-of-the-art GAN-based approach for synthetic data generation. It uses multiple tabular data-specific designs and outperforms many GAN-based baselines in terms of fidelity.
- *Probabilistic AutoRegressive (PAR)* [32]: A multivariant machine learning approach for sequential data generation. It has the ability to generate synthetic data with the time-series association learned from the real data.

All the baselines are trained by packet-level traffic. For the tabular approaches (GC, CTGAN), the dataset of a device is transformed to CSV tabular data by treating each packet as a row and extracting the same packet fields of IoTGemini as columns. For the sequential approach (PAR), the dataset of a device is firstly split by 5-tuple flows, and then the packets of a flow are segmented into multivariant sequences, and every packet consists of the same fields as the tabular data. All the data are properly normalized following the requirement of each baseline method. Note that the training dataset of the baselines will not be split by behaviors, which is the effort only made by the Device Modeling Module of our framework.

Downstream Tasks. To evaluate the usability of the synthetic traffic, we demonstrate three ML/DL-based downstream traffic analysis tasks:

- *Network Intrusion Detection System (NIDS)*: The NIDS uses traffic features to classify network traces into benign traffic and malicious traffic by supervised learning algorithms. We employ three ML-based approaches used in [2], including Linear Regression (LR), Decision Tree

(DT) and Random Forest (RF), and one DL-based approach used in [53], the bidirectional LSTM (BLSTM).

- *Anomaly Detection System (ADS)*: The ADS is typically based on unsupervised learning algorithms to detect deviation from the normal status like attacks and network failure. We employ three ML-based approaches used in [2], including Local Outlier Factor (LOF), Cluster-based Local Outlier Factor (CBLOF) and Isolation Forest (IF), and one DL-based approach used in [6], the 1D-CNN autoencoder (CNN-AE).
- *Device Fingerprinting (DF)*: The DF helps the network administrator identify the types of devices connected within its network, which is a supervised learning task. We employ one ML-based approach called DarkSide [54] and one DL-based approach called HomeMole [8].

To conduct experiments on these tasks, data alignment is necessary as the format of synthetic traffic differs from the input of these models. Hence, we write a script for feature extraction that can convert both the packet-level data and sequential data to the required input. For the ML models of these approaches, the input is tabular data of flow statistics, including mean, maximum, minimum, sum, standard deviation and variance of packet lengths and inter-arrival time. For the DL models, the input is packet-level sequential data similar to the output of IoTGemini and PAR, which are multivariant sequences of packet fields.

Metrics. We consider two categories of metrics respectively for the fidelity of data distribution and the usability to downstream tasks. For the evaluation of fidelity (i.e., similarity to real data), three metrics are used, all of which suggest better quality of synthetic data by higher scores:

- *Field Similarity*: It describes the overall distribution similarity of each packet field between real and synthetic data. Numerical fields (e.g., packet length) are compared by the Kolmogorov–Smirnov statistic, which is the maximum difference between the CDFs of real and synthetic data. Categorical fields (e.g., L4 protocol) are compared by the Total Variation Distance (TVD) as Equation 17 formulates, where ω is a possible value for a field Ω, and $\text{freq}_R(\omega)$ and $\text{freq}_S(\omega)$ represent the frequency of value ω in real data and synthetic data. The average value of the similarity of each field is calculated as the final score, which ranges between 0 and 1.

$$1 - \frac{1}{2} \sum_{\omega \in \Omega} |\text{freq}_R(\omega) - \text{freq}_S(\omega)| \tag{17}$$

- *Pair Trend*: It reflects how a pair of fields vary in relation to each other (e.g., direction and inter-arrival time). For a pair of the fields Ω_1 and Ω_2, it is calculated by Equation 18, where $\text{corr}_R(\Omega_1, \Omega_2)$ and $\text{corr}_S(\Omega_1, \Omega_2)$ is the Pearson correlation coefficient of the field pair in real data and synthetic data. The final score is calculated by the average value for all possible combinations of field pairs, which ranges between 0 and 1.

$$1 - \frac{1}{2} |\text{corr}_R(\Omega_1, \Omega_2) - \text{corr}_S(\Omega_1, \Omega_2)| \tag{18}$$

- *Sequentiality*: It describes the difficulty of telling apart real data from synthetic data that represent an ordered multivariant sequence of packet fields. We utilize the tool implemented by the SDMetrics library [55]. It first creates an augmented table with both real and synthetic data, and adds an extra column to keep track of whether each sequence is real or synthetic. Then it splits the augmented data into multiple training and validation sets. An LSTM classifier is trained on the training split and predicts whether each sequence is real or synthetic on the validation set. The final output is one minus the classification accuracy; a higher value close to 0.5 indicates better sequentiality of synthetic data.

From the perspective of data format, the field similarity and the pair trend evaluate the quality of synthetic traffic as independent packets, and the sequentiality is to tell the sequential fidelity of synthetic traffic as sequential data.

For the evaluation of usability, we use the relative error of F1 score for the tasks of network intrusion detection and anomaly detection. It is calculated by

$$\frac{|\text{F1}_R - \text{F1}_S|}{\text{F1}_R} \tag{19}$$

where F1_R is the F1 score of models trained by real data and F1_S is the F1 score of models trained by synthetic data. A lower difference indicates better usability of the synthetic data. The F1 score is a tradeoff metric between precision and recall, and is suitable for the evaluation of models under imbalanced scenarios like attack detection. For device fingerprinting, we use identification accuracy as the metric.

B. Effectiveness of Device Modeling

The modeling result of the devices in our IoT testbed is shown in Table III. Among the 19 devices, 12 devices have more than 3 functions and 8 devices have more than 5 BPs, which coincides with the trend of IoT being more functional nowadays and in future. In particular, cameras and audio devices possess more functions and BPs than other devices. As a typical type of smart devices, the increasing functionalities promote their usability in various scenarios, meanwhile highlighting the need for modeling devices to understand their behaviors. For appliances like lights and plugs, we do not observe any persistent functions on them (Pers. in Table III), as their functions are usually limited to remote control that is a transient function (Trans. in Table III).

To better understand the relations between functions and behaviors, we draw a Sankey diagram (Fig. 5) that illustrates the bipartite graph of the TP-Link camera as an example. On the left side are the functions including two persistent functions (“watch”, “mov-detect”), two transient functions (“photo”, “rotate”) and the idle state. On the right are the seven BPs. The width of the flows indicates packet numbers collected in 24 hours, when each function is triggered by 20 times and each persistent function runs for one minute per trigger, and keeping the device idle for the rest of the time. We observe that one function can be associated with multiple BPs. It means that the execution of a function probably needs the assistance of multiple services. Besides, it shows that the data

TABLE III
DEVICE MODELING OF OUR IOT TESTBED.

No.	Device Type	Vendor	Functions Trans.	Functions Pers.	BPs	Relation
1	power strip	Honyar	1	0	10	12
2	plug	Xiaomi	2	0	5	8
3	plug	Gree	1	0	3	6
4	light	Wiz	2	0	2	5
5	camera	Xiaomi	6	2	11	22
6	camera	Skyworth	1	2	4	9
7	camera	TP-Link	2	2	7	12
8	camera	HiChip	2	1	4	8
9	camera	Mercury	2	2	9	12
10	camera	EZVIZ	2	2	4	13
11	camera	Philips	2	1	2	10
12	camera	360	3	2	6	15
13	hub	Xiaomi	1	1	5	7
14	hub	Aqara	1	1	1	3
15	hub	Huawei	2	1	3	7
16	hub	Gree	1	0	1	3
17	audio	MiAI	5	1	13	25
18	audio	Baidu	2	1	4	10
19	audio	Alibaba	3	2	2	14

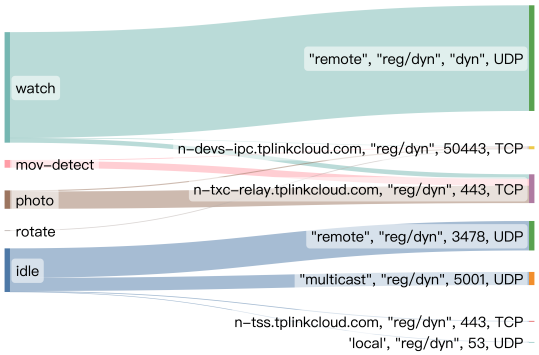
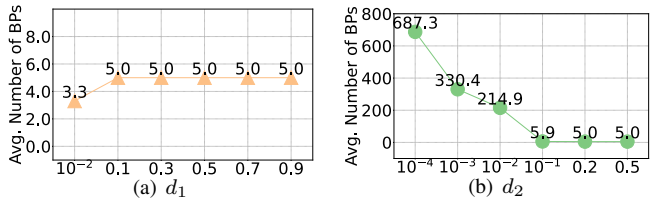


Fig. 5. An illustration of device modeling on TP-Link camera.



(a) d_1 (b) d_2

Fig. 6. Influence of two threshold hyperparameters in Behavior Profiling.

proportion of different functions and BPs is highly imbalanced and susceptible to user habits. For example, the device traffic pattern can be greatly changed if a user does not use the companion app to watch. This suggests the difficulty of using existing methods to generate custom traffic data, and highlights the need for dissecting device functions and behaviors as our framework.

We also evaluate the influence of two hyperparameters on behavior profiling, i.e., the threshold d_1 of spatial correlation and the threshold d_2 of temporal correlation for statistical constraint. As shown in Fig. 6, we find that the number of BPs converges when d_1 and d_2 are over 0.1. Since our knowledge-based constraint provides prior information on clustering, this makes the statistical-based constraint easier to satisfy, thus reducing the difficulty of hyperparameter selection.

For the other datasets, though we cannot obtain their functions and relations due to lack of their physical devices, we can still obtain the BPs using their traffic data. In Table IV, we compare the granularity of BPs to 5-tuples which is used as a unit of traffic in [31]. It can be seen that the number of BPs is significantly smaller than the number of 5-tuples for all the datasets and devices. It means that, compared to traffic generators built by each 5-tuple, the number of generators in IoTGemini can be reduced by over 99%, which significantly reduces the complexity of the overall system.

Therefore, the experiment shows that IoTGemini achieves effective device modeling, including network behavior profiling with proper granularity and the relations between device functions and network behaviors. It also lays the foundation for customized and high-quality traffic generation.

C. Fidelity of Data Distribution

To verify the validity of choosing the LSTM as the main component in the traffic generator of IoTGemini (i.e., PS-GAN), we first conduct an ablation experiment by replacing the LSTM with other neural networks and keeping the training process unchanged. The candidates include the Convolutional Neural Network (CNN) and the Multilayer Perceptron (MLP). We use the `CN` traffic dataset for the experiment. As Fig. 7 shows, the generator using the LSTM produces the best quality of synthetic traffic by all the metrics. We attribute the result to its superior ability to learn the sequential relationships between the packets of traffic, which is an important feature for many downstream tasks.

Subsequently, we compare the overall quality of synthetic traffic generated by IoTGemini with the real traffic of the four datasets using the metrics. The quantitative evaluation results in each dataset are depicted in Fig. 8. We find that IoTGemini shows good results on most of the datasets and metrics. Specifically, IoTGemini greatly outperforms the baselines by the metric of sequentiality, which highlights its advantage of using the sequential model and data format for traffic generation. Among the baselines, CTGAN and GC barely have any ability to generate sequential data; PAR as a sequential data generation approach exhibits relatively good sequentiality, but the result is still far inferior to the result of IoTGemini. The experiment result demonstrates that IoTGemini can generate

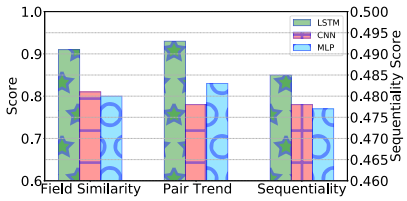


Fig. 7. Overall quality of synthetic traffic compared to real traffic using different components in PS-GAN.

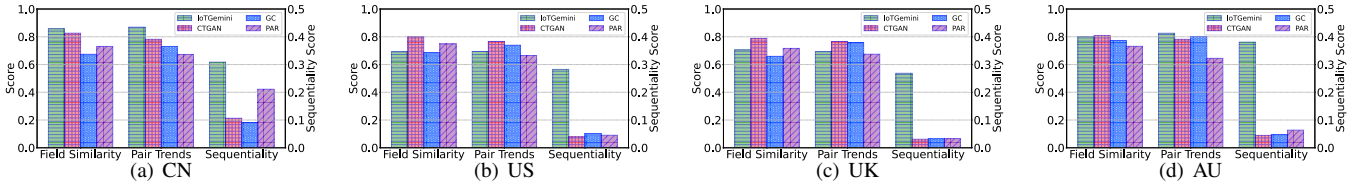


Fig. 8. Overall quality of synthetic traffic compared to real traffic in four datasets.

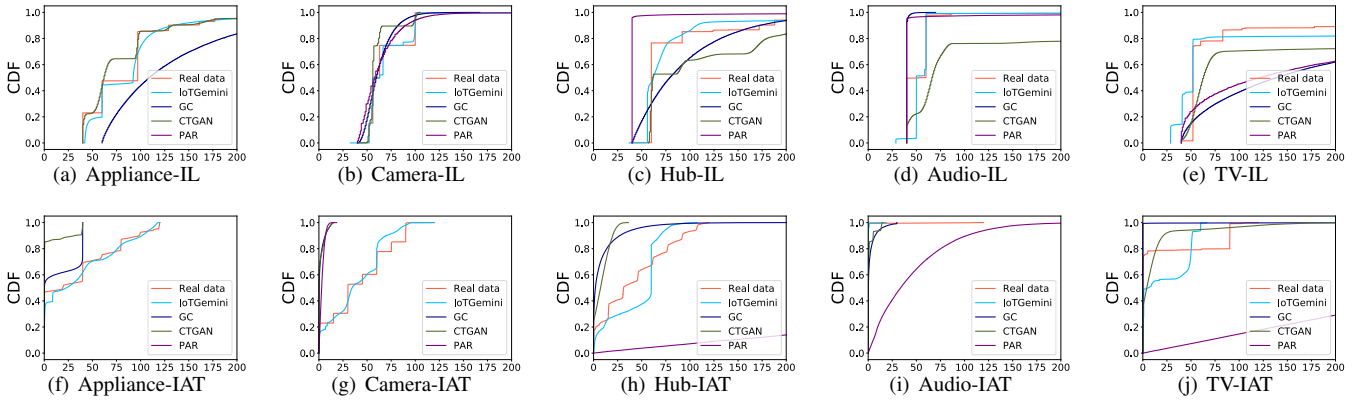


Fig. 9. Distribution of real and synthetic packet lengths (IL) and inter-arrival time (IAT) in different types of devices.

TABLE IV
COMPARISON BETWEEN THE NUMBER OF 5-TUPLES AND BPS.

Dataset	Device Type	Device Number	Average 5-tuples	Average BPs	Reduce By
CN	Appliance	4	687.3	5.0	99.27%
	Camera	8	24294.13	5.75	99.98%
	Hub	4	68.25	2.5	96.34%
	Audio	3	49431.67	6.33	99.99%
US	Appliance	16	165.08	4.72	97.14%
	Camera	9	1713.3	5.36	99.69%
	Hub	7	1360.0	2.8	99.79%
	Audio	4	7881.75	29	99.63%
	TV	5	1965.5	38.75	98.03%
UK	Appliance	11	98.77	1.7	98.28%
	Camera	7	480.66	4.42	99.08%
	Hub	6	2153.5	3.2	99.85%
	Audio	5	15111.8	23.2	99.85%
	TV	4	5050.0	24.66	99.51%
AU	Appliance	3	816.7	2.67	99.67%
	Camera	5	7548.0	16.0	99.79%
	Hub	2	4399.5	3.0	99.93%
	Audio	2	12763.0	10.0	99.92%

synthetic traffic with high fidelity from the view of both packet-level data and sequential data.

Besides the overall quality, we are also curious about the fidelity of each important packet field. We primarily explore two packet fields – IP packet length (IL) and inter-arrival time (IAT), as these two fields are remarkably used by almost every traffic analysis model (e.g., [2], [3], [5], [6], [8], [9], [21]).

We measure the field values on the real traffic of different device types (appliance, camera, hub, audio, TV) and the synthetic traffic of these devices, and illustrate their cumulative distribution functions (CDF) for qualitative analysis. The

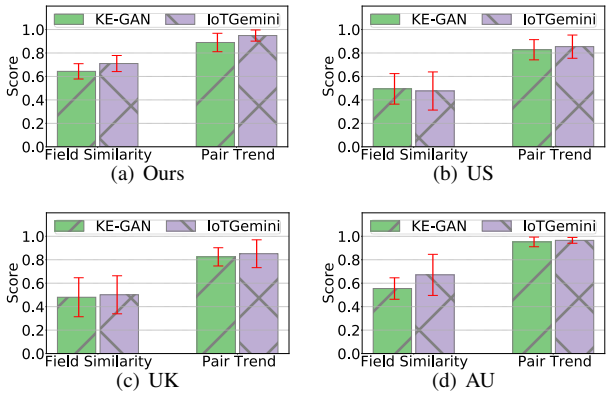


Fig. 10. Quality of synthetic aggregate features compared to KE-GAN [36].

results are in Fig. 9. It can be seen that the synthetic traffic of IoTGemini (curve in light blue) exhibits the most similar trend to the real traffic (curve in orange). In particular, we find a pattern of the real traffic that many CDF curves go alternately steep and flat. It means that many field values fall within different small ranges, implying the existence of multiple behaviors inside the device traffic. We observe that only IoTGemini can best mimic this pattern. In contrast, baselines such as GC and PAR cannot generate similar shapes of distribution but produce relatively smooth curves. It can be explained by the baselines failing to generate the variety of the distribution, while IoTGemini benefits from its device modeling that splits network behaviors to decrease the difficulty of learning the traffic patterns. This experiment again shows the high fidelity of the synthetic traffic generated by IoTGemini.

In addition to straightforward packet-level synthetic data fidelity assessment, we also evaluate the quality of aggregate

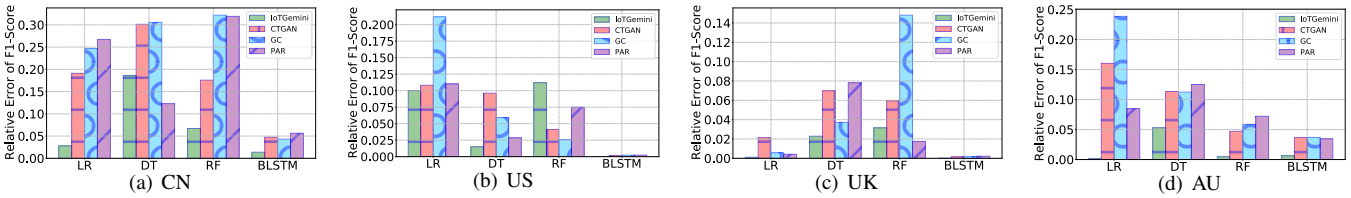


Fig. 11. Relative error of F1 score of network intrusion detection approaches trained by synthetic traffic and tested by real traffic, compared to the approaches trained by real traffic.

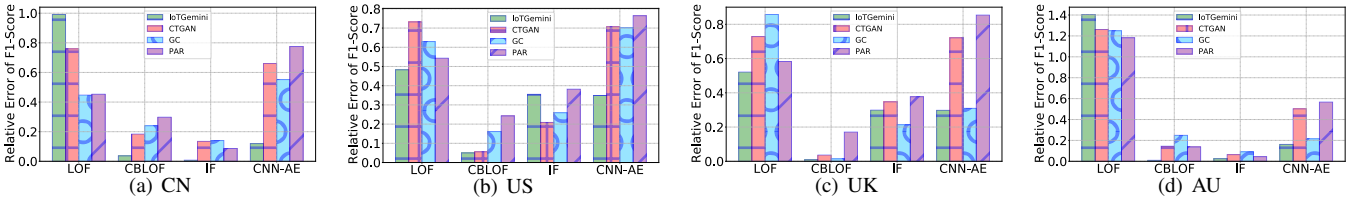


Fig. 12. Relative error of F1 score of anomaly detection approaches trained by synthetic traffic and tested by real traffic, compared to the approaches trained by real traffic.

traffic features post feature extraction. In this experiment, we employ the state-of-the-art Knowledge Enhanced GAN (KE-GAN) as our baseline method, as described in [36]. It is worth noting that KE-GAN does not generate packet-level data; instead, it produces three aggregate features for each IoT device, including total packet count within a flow, average packet length within a flow, and inter-arrival time. For comparative purposes with KE-GAN, we first utilize IoTGemini to generate packet-level traffic and subsequently compute the same aggregate features. As shown in Fig. 10, the quality of these aggregate features is compared with those of real traffic across four datasets. The results indicate that IoTGemini closely aligns with KE-GAN's performance and even exhibits superior performance on some datasets. It is important to note that KE-GAN directly learns from the aggregate features of real traffic, whereas IoTGemini only learns from packet-level data and then transforms synthetic data into aggregate features. The results demonstrate that IoTGemini maintains a high level of fidelity to real traffic even after further feature aggregation process that introduces additional information loss.

D. Usability to Downstream Tasks

To verify the usability of synthetic traffic to downstream traffic analysis tasks, we use the synthetic traffic of IoTGemini and the baselines to train the models of the tasks and use the real traffic to test the trained models. To demonstrate the versatility of generated packet-level traffic that can be converted into other data formats, for the models of the three downstream tasks in the experiment, we employ at least one ML-based approach that inputs flow-level statistics and one DL-based approach that inputs packet-level sequential data.

NIDS. Four supervised learning models are trained as binary classifiers to distinguish between IoT traffic (as benign traffic) and malicious traffic. The experiment results are shown in Fig. 11. IoTGemini outperforms the baselines for most of the datasets and the NIDS approaches. In addition, the superiority of IoTGemini to the baselines can be significant. For example,

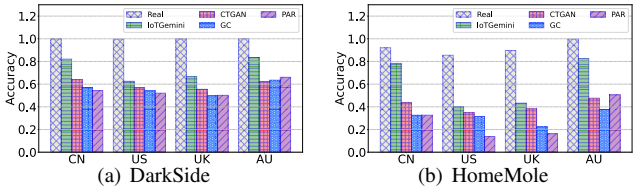


Fig. 13. Identification accuracy of device fingerprinting approaches trained by synthetic traffic and tested by real traffic.

in the AU dataset, the relative error of IoTGemini on the LR approach is only 1.45×10^{-3}, whereas the relative error of the baseline GC is much higher (0.239).

ADS. Four unsupervised learning models are trained using only the IoT traffic (as benign traffic), and the trained models output anomaly scores to tell the anomalous traffic. As shown in Fig. 12, the synthetic traffic of IoTGemini can train the models with the lowest relative error for most of the datasets and approaches. Note that the relative errors of ADS approaches are mostly higher than the relative errors of NIDS approaches. It is because some unsupervised learning models are more difficult to train than supervised learning models. For example, the F1 score of the LOF on the CN dataset is only 0.365 even trained by real data. Accordingly, the high relative errors obtained by the synthetic traffic are also reasonable. We find that IoTGemini achieves much lower relative error than the baselines for the CNN-AE approach. As it is a DL-based approach using packet sequences as input, the advantage of IoTGemini can be attributed to its ability to generate the sequential relationships among packets as the real traffic.

DF. One ML-based approach (DarkSide) and one DL-based approach (HomeMole) are trained to identify device types using the IoT traffic. As Fig. 13 illustrates, the approaches trained by the real data achieve the highest accuracy. As for the approaches trained by the synthetic traffic, IoTGemini outperforms other baselines for all the datasets, and its superiority is greater on HomeMole. Again, it suggests the compatibility of

IoTGemini with DL-based approaches that require the learning of temporal/spatial relationships. As the DL-based approaches to traffic analysis are becoming more popular for their better performance, the advantage of IoTGemini is also highlighted.

To sum up, the experiments demonstrate IoTGemini's high usability of synthetic traffic generation to facilitate various downstream tasks, and its good compatibility with different models and data formats used by the approaches of these tasks. At this point, all three questions raised at the beginning of this section are properly answered.

IX. Discussion and Future Work

In this section, we discuss the limitations and open questions of IoTGenimi. We also offer insights on the possible solutions and future directions for optimization.

Scope and generalization. This paper primarily concentrates on IP traffic, as IP-enabled devices hold a major share of the IoT market, especially for the emerging intelligent devices in many scenarios (e.g., smart homes, smart cities) [56]. In our future work, we expect to expand the scope of device modeling to a wider range of protocol stacks, such as Zigbee and Z-Wave. With these protocol stacks, a revised Device Modeling Module is required as their traffic cannot be described by 5-tuples. A possible solution could be constructing BPs using their specific description of traffic (e.g., the fields in NWK layer, APS layer and APL layer of Zigbee). The Traffic Generation Module can be easily generalized with just minor adjustments, considering the universal characteristics of data transmission by packets among the protocol stacks. For further study, research on generating higher-resolution traffic data is meaningful, such as in units of 100ms or 10ms to meet today's low-latency trend of IoT networking.

Function acquisition automation. In IoTGemini, we use a Bash script with ADB to automate the triggers of device functions. Writing the script can be labor-consuming as we need to beforehand explore the UI of IoT companion apps. The function acquisition could be incomplete, since our script is difficult to cope with multi-layer menus or text input UI components in the apps. In future work, to build a larger IoT testbed, we expect to address these two issues by exploiting some UI component recognition tools (e.g., OCR [57]). For open-source apps, the function acquisition of these apps can hugely benefit from automated code analysis tools.

Stability of modeling. As time goes by, a device may obtain connections to a new domain due to domain migration, which affects the completeness of BPs. A device may also exhibit new traffic patterns due to firmware update that changes the protocol, which affects the performance of synthetic traffic generators. To guarantee stability, one of our future work is to integrate the continuous learning technique with our IoTGemini framework for model updates. Besides, we are also curious if the latest time-series modeling approach Transformer [58] can achieve better stability on certain complex behaviors, given its attention structure and the larger number of parameters.

Security considerations. We assume the physical devices are located in a safe environment. In practice, many devices have to work in an open space where malware is likely to compromise the devices. The malicious behaviors may be propagated from the malware to the generated data, which is undesirable for downstream tasks. To address the security concern, we can deploy a traffic cleaning system before the downstream tasks to filter unwanted traffic data.

X. Conclusion

This paper proposes IoTGemini, a framework for the generation of IoT synthetic traffic, which consists of a Device Modeling Module and a Traffic Generation Module. The Device Modeling Module enables the lifecycle modeling of device functions and network behaviors. For each of the network behaviors, the Traffic Generation Module builds a GAN-based generator to produce synthetic traffic. Accordingly, IoTGemini supports the customized use of device functions and the generation of the corresponding traffic. Our evaluation shows that IoTGemini achieves great effectiveness in device modeling, high fidelity of synthetic traffic and remarkable usability in various downstream traffic analysis tasks.

References

[1] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "Iot SENTINEL: automated device-type identification for security enforcement in iot," in *37th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2017.

[2] Y. Wan, K. Xu, G. Xue, and F. Wang, "Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes," in *39th IEEE Conference on Computer Communications, INFOCOM*, 2020.

[3] R. Doshi, N. J. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops, SP Workshops*, 2018.

[4] L. Giaretta, A. Lekssays, B. Carminati, E. Ferrari, and S. Girdzijauskas, "Limnet: Early-stage detection of iot botnets with lightweight memory networks," in *26th European Symposium on Research in Computer Security, ESORICS*, 2021.

[5] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018.

[6] R. Li, Q. Li, Y. Huang, W. Zhang, P. Zhu, and Y. Jiang, "Iotensemble: Detection of botnet attacks on internet of things," in *27th European Symposium on Research in Computer Security, ESORICS*, 2022.

[7] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS*, 2019.

[8] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your smart home can't keep a secret: Towards automated fingerprinting of iot traffic," in *ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security*, 2020.

[9] X. Ma, J. Qu, J. Li, J. C. S. Lui, Z. Li, and X. Guan, "Pinpointing hidden iot devices via spatial-temporal traffic fingerprinting," in *39th IEEE Conference on Computer Communications, INFOCOM*, 2020.

[10] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy, "DEFT: A distributed iot fingerprinting technique," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 940–952, 2019.

[11] Q. Kong, F. Yin, R. Lu, B. Li, X. Wang, S. Cui, and P. Zhang, "Privacy-preserving aggregation for federated learning-based navigation in vehicular fog," *IEEE Trans. Ind. Informatics*, vol. 17, no. 12, pp. 8453–8463, 2021.

[12] J. Ren, D. J. Dubois, D. R. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach," in *Proceedings of the Internet Measurement Conference, IMC*, 2019.

[13] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[14] "Ns-3 network simulator," https://www.nsnam.org/

[15] K. V. Vishwanath and A. Vahdat, "Swing: realistic and responsive network traffic generation," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 712–725, 2009.

[16] S. Xu, M. Marwah, and N. Ramakrishnan, "STAN: synthetic network traffic generation using autoregressive neural models," *CoRR*, vol. abs/2009.12740, 2020.

[17] P. Wang, S. Li, F. Ye, Z. Wang, and M. Zhang, "Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using CGAN," in *Proceedings of IEEE International Conference on Communications, ICC*, 2020.

[18] J. Sommers, H. Kim, and P. Barford, "Harpoon: a flow-level traffic generator for router and network tests," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS*, 2004.

[19] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "Syngan: Towards generating synthetic network attacks using gans," *CoRR*, vol. abs/1908.09899, 2019.

[20] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Comput. Secur.*, vol. 82, pp. 156–172, 2019.

[21] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," in *27th Annual Network and Distributed System Security Symposium, NDSS*, 2020.

[22] A. Cheng, "Pac-gan: Packet generation of network traffic using generative adversarial networks," in *Proceedings of IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019.

[23] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A. Sadeghi, and A. S. Uluagac, "Peek-a-boo: i see your smart home activities, even encrypted!" in *WiSec '20: 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Linz, Austria, July 8-10, 2020*. ACM, 2020.

[24] Y. Wan, K. Xu, F. Wang, and G. Xue, "Iotathena: Unveiling iot device activities from network traffic," *IEEE Trans. Wirel. Commun.*, vol. 21, no. 1, pp. 651–664, 2022.

[25] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection based on meta-learning framework," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3540–3552, 2020.

[26] "Ostinato: Traffic generator for network engineers," https://ostinato.org.

[27] "Rude & crude," https://rude.sourceforge.net.

[28] "Seagull: an open source multi-protocol traffic generator," https://gull.sourceforge.net.

[29] "Azure iot device telemetry simulator," https://github.com/Azure-Samples/Iot-Telemetry-Simulator, 2004.

[30] H. Redžović, A. Smiljanić, and M. Bjelica, "Ip traffic generator based on hidden markov models," https://www.etran.rs/common/pages/proceedings/IcETRAN2017/TEI/IcETRAN2017_paper_TEI2_3.pdf, 2017.

[31] Y. Yin, Z. Lin, M. Jin, G. Fanti, and V. Sekar, "Practical gan-based synthetic IP header trace generation using netshare," in *SIGCOMM '22: ACM SIGCOMM 2022 Conference*, 2022.

[32] K. A. Zhang, N. Patki, and K. Veeramachaneni, "Sequential models in the synthetic data vault," *CoRR*, vol. abs/2207.14406, 2022.

[33] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *CoRR*, vol. abs/1406.2661, 2014.

[34] H. Nguyen-An, T. Silverston, T. Yamazaki, and T. Miyoshi, "Generating iot traffic: A case study on anomaly detection," in *26th IEEE International Symposium on Local and Metropolitan Area Networks, LANMAN 2020, Orlando, FL, USA, July 13-15, 2020*. IEEE, 2020.

[35] M. R. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative deep learning for internet of things network traffic generation," in *25th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2020, Perth, Australia, December 1-4, 2020*. IEEE, 2020, pp. 70–79.

[36] S. Hui, H. Wang, Z. Wang, X. Yang, Z. Liu, D. Jin, and Y. Li, "Knowledge enhanced GAN for iot traffic generation," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*. ACM, 2022.

[37] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional GAN," in *Proceedings of Advances in Neural Information Processing Systems*, 2019.

[38] "Recommendations for transport-protocol port randomization," https://www.rfc-editor.org/info/rfc6056.

[39] "openhab," https://www.openhab.org/, 2023.

[40] "Apple home," https://www.apple.com/home-app/, 2023.

[41] "Android debug bridge," https://developer.android.com/studio/command-line/adb.

[42] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium*, 2008.

[43] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2010.

[44] "Manufacturer usage description specification (rfc 8520)," https://datatracker.ietf.org/doc/rfc8520/, 2019.

[45] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman, "Clear as MUD: generating, validating and applying iot behavioral profiles," in *Proceedings of the 2018 Workshop on IoT Security and Privacy, IoT S&P@SIGCOMM*, 2018.

[46] D. P. Bertsekas and R. G. Gallager, *Data Networks*. Prentice Hall, 1992.

[47] "Cisco systems netflow services export version 9," https://www.rfc-editor.org/rfc/rfc3954, 2004.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[49] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *CoRR*, vol. abs/1701.07875, 2017.

[50] V. Bezerra, V. Turrisi da Costa, R. Martins, S. Barbon, R. Miani, and B. Bogaz ZarpelÃ£o, "Providing iot host-based datasets for intrusion detection research," in *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, 2018.

[51] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Gener. Comput. Syst.*, 2019.

[52] D. Tjøstheim, H. Otneim, and B. Støve, "Chapter 5 - local gaussian correlation and the copula," in *Statistical Modeling Using Local Gaussian Approximation*. Academic Press, 2022, pp. 135–159.

[53] C. D. McDermott, F. Majdani, and A. Petrovski, "Botnet detection in the internet of things using deep learning approaches," in *2018 International Joint Conference on Neural Networks, IJCNN*, 2018.

[54] A. M. Hussain, G. Oligeri, and T. Voigt, "The dark (and bright) side of iot: Attacks and countermeasures for identifying smart home devices and services," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage - SpaCCS 2020 International Workshops*, 2020.

[55] *Synthetic Data Metrics*, DataCebo, Inc., 10 2022, version 0.8.0. [Online]. Available: https://docs.sdv.dev/sdmetrics/

[56] "Internet of things (iot) market size forecast, 2022-2029," https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307, 2022.

[57] "iflytek: Optical character recognition," https://global.xfyun.cn/products/wordRecg.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.