## GENERAL INSTRUCTION

- **Authors: When you submit your corrections, please either annotate the IEEE Proof PDF or send a list of corrections. Do not send new source files as we do not reconvert them at this production stage.**
- **Authors: Carefully check the page proofs (and coordinate with all authors); additional changes or updates WILL NOT be accepted after the article is published online/print in its final form. Please check author names and affiliations, funding, as well as the overall article for any errors prior to sending in your author proof corrections. Your article has been peer reviewed, accepted as final, and sent in to IEEE. No text changes have been made to the main part of the article as dictated by the editorial level of service for your publication.**
- **Authors: Per IEEE policy, one complimentary proof will be sent to only the Corresponding Author. The Corresponding Author is responsible for uploading one set of corrected proofs to the IEEE Author Gateway.**

## QUERIES

Q1. Author: Please confirm or add details for any funding or financial support for the research of this article.
Q2. Author: Please specify the name of the corresponding author.
Q3. Author: Please provide page range for Refs. [5], [15], and [28].
Q4. Author: Please provide the publication year for Refs. [7] and [26].

# Detecting Smart Home Device Activities Using Packet-Level Signatures From Encrypted Traffic

Mohammad Shamim Ahsan, Md. Shariful Islam, Md. Shohrab Hossain, *Member, IEEE*, and Anupam Das, *Senior Member, IEEE*

Anupam Das is the corresponding author

***Abstract*—Despite the significant benefits of the widespread adoption of smart home Internet of Things (IoT) devices, these devices are known to be vulnerable to active and passive attacks. Existing literature has demonstrated the ability to infer the activities of these devices by analyzing their network traffic. In this study, we introduce a packet-based signature generation and detection system that can identify specific events associated with IoT devices by extracting simple features from raw encrypted network traffic. Unlike existing techniques that depend on specific time windows, our approach automatically determines the optimal number of packets to generate unique signatures, making it more resilient to network jitters. We evaluate the effectiveness, uniqueness, and correctness of our signatures by training and testing our system using four public datasets and an emulated dataset with varying network delays, verifying known signatures and discovering new ones. Our system achieved an average recall and precision of 98-99% and 98-100%, respectively, demonstrating the effectiveness and feasibility of using packet-level signatures to detect IoT device activities.**

***Index Terms*—Network traffic analysis, packet-based signature.**

## I. Introduction

THE use of Internet of Things (IoT) devices (e.g., light sensors, cameras, door locks, thermostats, etc.) has increased rapidly in recent years. IoT provides a flexible and scalable platform that can support various applications within a home setting, e.g., enhance security, enable eldercare or childcare, save energy, and automate appliances [35]. However, IoT devices also create new attack vectors. For example, simply knowing what devices reside inside a household can pose both security and privacy risks. Knowing that there is a smart lock can enable an adversary to launch targeted attacks, e.g., exploit unpatched vulnerabilities. Similarly, knowing that there is a sleep monitoring device inside the home can reveal the sleeping patterns of residents.

While most IoT devices encrypt traffic using standard protocols such as WPA2, such encryption only hides the payload, that is, the contents of the exchanged messages or commands. However, the related meta-data (e.g., packet lengths, traffic rate) of the network traffic still leaks some information about the messages exchanged [38]. Existing works in this domain have looked at extracting statistical features from traffic metadata to detect device events [1], [10], [11], [37], but such approaches are prone to background traffic noise and thus harder to generalize across varying settings [4]. Moreover, most existing passive inference techniques can only identify the device type and whether there is device activity (i.e., an event), but not the exact type of event or command [2], [3], [22], [23], [24], [25]. The state-of-the-art work on packet-level signatures for IoT devices is PINGPONG [28], which relies on the packet arrival time window to detect device activities. This time-window dependency can result in inaccurate detection whenever the traffic rate significantly changes; because the number of packets considered in a specific time interval for matching signatures will be drastically more or less than that considered during training. Also, such time delays can vary significantly depending on the underlying internet connection. Furthermore, for IoT devices that have more than two forms of device status (e.g., STANDBY, ACTIVE, QUICKRUN, STOP), PINGPONG considers separate events as binary type and trains them independently, rather than training all the events of the device at a time, resulting in additional overhead.

Our work aims to address the existing limitations of state-of-the-art techniques to determine user activities on IoT devices solely using encrypted network traffic. In particular, we seek to answer the following questions: *RQ1: Can we develop an activity detection system that is independent of time-window?* We develop a methodology where we depend on a threshold of packets rather than time-window for detecting events in any network traffic. *RQ2: How can we generate packet-level signatures for devices with multi-type events without prior knowledge about a device's behavior?* We analyze how to automatically construct signatures from devices with multi-type events using different features extracted from the network traffic. *RQ3: How well do the generated signature(s) of a specific device maintain uniqueness and correctness across independent datasets?* We evaluate to what extent the signatures of a device are unique and generalizable across various datasets collected from different locations and under different settings.

In summary, we introduce a packet-based signature generation and detection system that does not consider a temporal threshold and can efficiently handle both *binary* and *multi-type* events. Our system overcomes these limitations as we

Manuscript received 1 August 2023; revised 28 June 2024; accepted 1 July 2024.

Q1 Mohammad Shamim Ahsan, Md. Shariful Islam, and Md. Shohrab Hossain are with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka 1000, Bangladesh (e-mail: shamim19119@gmail.com; sharifulislam08031998@gmail.com; mshohrabhossain@cse.buet.ac.bd).

Q2 Anupam Das is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695 USA (e-mail: anupam.das@ncsu.edu).

Digital Object Identifier 10.1109/TDSC.2024.3424299

experiment with a diverse range of smart home devices and use only the packet length and direction for a specific number of packets, simplifying the detection technique and achieving higher recall and precision rates. We make the following contributions:

- To generate and detect packet-level signatures, we establish a threshold on the number of packets instead of relying on time windows. Time window-based methods are less effective across different datasets as they can be easily impacted by traffic rate changes. Our approach is more resilient to temporal fluctuations caused by network jitters, as demonstrated through experiments on real-world datasets.
- To the best of our knowledge, we have obtained the highest average recall and precision of 98−99% and 98−100%, respectively, for any packet-level signature detection system to date. We have also open-sourced our codebase.[1]
- Our automated process captures *new signatures* from existing datasets, verified through cross-data validations. This method systematically generates signatures for both binary-type and multi-type events without prior knowledge of the device's behavior.

## II. Background and Related Work

### A. Key Components of IoT-Based System

A typical IoT system has four key components that work together to provide the desired functionalities. An essential component of an IoT system is sensors or end devices that observe and capture environmental and physical changes. All the collected data can be used in various domains and might have various degrees of complexity. The second key component of IoT is connectivity. The collected data needs to be sent to a cloud-based infrastructure so that it can be processed and analyzed. Based on the scale of the implementations, IoT components can be connected over LANs, MANs, or WANs. It can also be connected through Wi-Fi, Bluetooth, telephone networks like LTE (popularly known as 4 G Network), or light-based technologies like Li-Fi (where light is used as a mode of communication to maintain interconnections) [7]. The gathered data from sensors are transmitted over the internet using one of these technologies. The analytic component starts functioning once the data is collected and delivered to the cloud. IoT analytics are utilized here to get a sense of the gathered data and to prevent it from getting corrupted. Last but not least, the main internal key component of IoT is the user interface which allows humans to interact with IoT devices and systems. The collected information from the previous components can be made available to users either in report format or in the form of some actions like triggering an alarm, notifying them through emails or texts, etc.

### B. Data Communication Model for IoT

There are three different methods of communication between an IoT device and the backend cloud. First, an IoT device can directly communicate with the cloud (i.e., Device ↔ Cloud). Second, the companion mobile app can exchange information with the IoT device directly (i.e., Phone ↔ Device). Lastly, the companion mobile app can communicate with the cloud (i.e., Phone ↔ Cloud). IoT manufacturers typically create companion mobile apps to configure, control, and interface with their corresponding devices. Therefore, data from the IoT device can also reach the IoT cloud via the companion app installed on a smartphone [36].

### C. Related Works

There is a large body of work in the network measurement community that uses traffic analysis to fingerprint applications [5], [9], [12], identify anomalies [13], [21], attacks [32], or malware [31]. In recent years, there have been multiple studies to characterize traffic from IoT devices and develop techniques to fingerprint their activities.

Hafeez et al. propose *IoTGUARD* [10], a self-adaptive semi-supervised learning-based classification scheme for real-time activity detection of IoT devices in edge networks. It predicts malicious traffic based on the network activity of the device generating the traffic. *Peek-a-Boo* [1] introduces a *multi-stage privacy attack* utilizing machine-learning approaches for detecting and identifying the types of IoT devices, their states, and ongoing user activities by only passively sniffing the network traffic. *IOT-KEEPER* [11] detects malicious IoT network activity using a combination of fuzzy k-means clustering and a fuzzy interpolation scheme for online traffic analysis at the edge gateways. *HomeSnitch* [37] classifies IoT device semantic behavior such as heartbeat, firmware check, and motion detection using statistics derived from the entire client-server dialog. *PINGPONG* [28] explores the sequential and directional "ping/pong" behavioral patterns in IoT data communication. Interestingly, the most important feature used in HomeSnitch is the average number of bytes sent from the IoT device to the server per turn, whereas PINGPONG uses packet lengths and directions of individual requests (and replies) to uniquely identify device events.

A recent paper by Ren et al. [15] presents a large-scale and multidimensional analysis of information exposure from IoT devices and reveals how these devices operate differently due to different privacy regulations in the US and U.K.. The paper also presents a classifier to infer event types spanning many device categories. Researchers have also used DNS queries to infer IoT devices [23], [30]. They also model traffic characteristics to infer device-level activities. Sivanathan et al. [2], [3] also leverage network traffic and uniquely build a multi-layer model to fingerprint IoT devices. Ahmed et al. [4] present the largest to-date study of IoT device fingerprinting in which they consider multiple factors such as fingerprinting across time, region, and different datasets and under different constraints. Others have utilized data from different layers of the system stack to identify IoT devices [18], [34].

*Limitations of Existing Works:* Most event inference techniques rely on machine learning [1], [2], [3], [10], [17] or statistical analysis of traffic time series [6], [11], [15], [37].

[1] https://github.com/MdShamim097/Packet-based-IoT-Event-Detection/

TABLE I
COMPARISON WITH RELATED WORK ON IOT DEVICES

| Work | Number of features used | Aggregated data used | Granularity of detection | | Dataset used (geographic origin) | Jitter resilient | Multi-event detection | Discovering new signatures | Computational overhead † |
|---|---|---|---|---|---|---|---|---|---|
| | | | Device-activity* | Device make-&-model◇ | | | | | |
| IoTGUARD [10] | 39 | ✓ | × | × | 2 (US, UK) | × | × | × | — |
| Peek-a-Boo [1] | 197 | × | ✓ | ✓ | 2 (US, IT) | × | × | × | — |
| IoT-KEEPER [11] | 38 | × | ✓ | × | 3 (US, NL) | × | × | × | — |
| Apthorpe et al. [25] | 8 | × | ✓ | ✓ | 1 (US) | × | × | × | — |
| Ren et al. [15] | 14 | ✓ | ✓ | × | 2 (US, UK) | × | ✓ | × | — |
| Sivanathan et al. [3] | 4 | ✓ | × | ✓ | 1 (AU) | × | × | × | — |
| HomeSnitch [37] | 13 | ✓ | ✓ | ✓ | 2 (US) | × | × | × | — |
| PINGPONG [28] | 2 | × | ✓ | ✓ | 4 (US, AU) | × | × | × | $\mathcal{O}(xn)$ |
| Our work | 2 | × | ✓ | ✓ | 4 (US, AU) + jitter dataset | ✓ | ✓ | ✓ | $\mathcal{O}(x)$ |

* Inferring individual device-level activity; ◇ Individual devices (different make and model); † $x$=number of devices, $n$=number of event types for each device; '—' refers to N/A as they use an ML-based approach rather than a packet-level signature.

These techniques have different limitations as they use different approaches. *IoTGUARD* [10] cannot detect the event of a device; rather, it can only tell whether an attack is going on or not. Besides, the aggregated features used here are not always appropriate for all scenarios; for instance, the total amount of data transferred and the total number of connections made depend on higher-level user activity rather than device activity. Unsupervised learning techniques may be hard to interpret, especially for large feature sets (e.g., 197 features in *Peek-a-Boo* [1]). *HomeSnitch* [37] lacks resistance to traffic fluctuations. Moreover, it uses statistics (average, min, max) derived from the entire client-server dialog, which may lead to false positives at a large scale. Signature generation and detection in *PINGPONG* [28] depends on time-window, which may result in incorrect detection if the traffic rate slows down or speeds up. *IOTKEEPER* [11] uses statistical approaches that may reduce clustering performance when dealing with a large number of features.

Moreover, these machine-learning approaches have limitations in differentiating event types (e.g., distinguishing ON from OFF). Especially, [2], [3], [17] use features like sleep time, flow volume-duration, average packet size, peak-to-mean ratio, DNS queries, and cipher suites, or uses source port, destination port, direction, payload, window size and timestamps, which can only identify the IoT devices; not the specific events of those devices. Statistical analysis approaches depending on time series also lack resistance to traffic fluctuations [6], [11], [15], [37]. A series of papers by Apthorpe et al. [22], [23], [24], [25] use traffic volume-based signatures to infer IoT device activity, but cannot always determine the exact type of the event. Arguably, by observing changes in traffic rates, it is only possible to identify whether any user interaction occurs or not; but not which kind of interaction has occurred.

*Distinction from Prior Work:* Our work is inspired by the aforementioned work, especially PINGPONG [28], which detects user activity of IoT devices using packet-level signatures. Our primary and differentiating goal is to overcome the limitations of existing works by implementing a "packet-based signature generation and detection system," which depends on packet counts instead of time-window, making it more resilient against time delays and network jitters. We also focus on handling multi-type events rather than only binary (ON/OFF) events as analyzed by PINGPONG [28]. Through comprehensive evaluation, we see that our proposed approach performs better in detecting activities of IoT devices and sensors, as described in Section IV. Moreover, we discover some new signatures that are not only different from the existing ones but also smaller in size, indicating the efficiency of our packet-based approach. Furthermore, the development of the multi-event detection technique makes our system more resource-efficient since it requires only $\mathcal{O}(x)$ computational overhead; whereas PINGPONG requires $\mathcal{O}(xn)$, where, $x$ and $n$ are the number of devices and the number of event types for each of the devices, respectively. Table I provides an in-depth comparison with related works, highlighting the contributions of our work over existing studies.

## III. PROPOSED APPROACH

In this section, we describe our system architecture, methodologies, algorithms, and heuristics for generating signatures.

### A. Threat Model

Our setup accounts for a passive adversary capable of monitoring network traffic from smart home devices. We consider two types of adversaries: a *WAN sniffer* and a *Wi-Fi sniffer*, as seen in previous works [28]. The WAN sniffer observes network traffic between the home router and the ISP network (or beyond), while the Wi-Fi sniffer monitors encrypted IEEE 802.11 wireless traffic. The WAN sniffer can inspect IP headers but lacks device MAC addresses for traffic identification. The Wi-Fi sniffer lacks the WPA2 key and can only access clear text information, including MAC addresses, packet lengths, and timing data. Both adversaries are aware of the smart home device type they want to target and passively monitor. Thus, they can train the system on a similar device offline, extract its signature, and use it to detect the targeted device's signature from the monitored traffic. It is assumed that the devices encrypt their communication, preventing adversaries from accessing clear-text communication.

### B. Architecture

In this paper, we focus on improving the state-of-the-art work, PINGPONG [28], to automatically and accurately unveil user activities from smart home network traffic logs. Our system includes three key stages: i) Input processing, ii) Training, and iii) Activity detection. In the first stage, we take necessary annotated inputs, such as a device's event types and triggered events. Then, the training stage begins, where we analyze network traffic, extract features, make a pair of relevant packets,

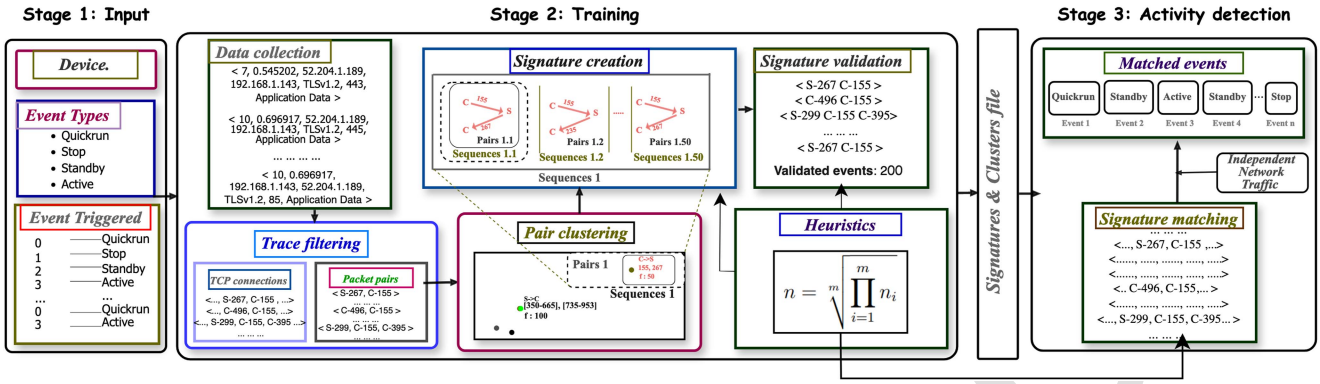


Fig. 1. Our System Architecture. At first, a device's event types and triggered events are taken as input. Then, we collect network traffic, extract features, and make a pair of relevant packets. After that, we generate signatures that consist of a particular number of ordered sequences of packet pairs. Next, we validate each event's signatures. At last, we test our system to detect activity on independent network traffic. In the last two stages, we use our heuristic.

generate signatures using some particular ordered packet pairs, and cross-validate our generated signatures. In the final stage, we test our system on *independent* network traffic to detect the activities of a given device. In the last two stages, we use our proposed heuristic. The architecture of our system is depicted in Fig. 1.

❶ *Input Processing:* At first, we take a device's event types and trigger events of that type as input. For convenience, we consider each event type as an integer at the time of training, as illustrated in Fig. 1.

❷ *Training:* We utilize the PINGPONG dataset [28] to generate and test packet-level signatures. We will also demonstrate generalizability using other public datasets [3], [14], [27].

*Trace Filtering:* We filter the collected raw training set (a set of `pcap` files and event-based timestamps) to discard unrelated traffic. We discard the packets where neither the source nor destination IP matches. Also, the packets beyond a number of packets, $n$ (a threshold we described later on), after each timestamped event is discarded.

*Pair Clustering:* In this step, we separate relevant packet pairs (i.e., those that consistently occur after an event) from irrelevant ones. Since we do not know in advance the packet lengths in pairs, we use an unsupervised learning algorithm, called DBSCAN [19] that makes clusters of similar data points, where the similarity is determined by the euclidean distance between the points. In this algorithm, we mark some points as core points if they have a certain amount of points (amount may be number or percentage) within a particular distance d from them. If two core points are situated within d from each other, they will be in the same cluster. Finally, the non-core points (i.e. leftover points) will join any cluster if that particular cluster is at most d distance from them. However, the points without any cluster are treated as noise or outliers that can be ignored.

*Signature Creation:* Signatures in this paper are identified based on only two parameters: *packet-length* and *direction*. To create a signature, we concatenate packet pairs in the clusters to reassemble the longest packet sequences possible. Packet pairs in clusters $x$ and $y$ are concatenated if, and only if, for each packet pair $p_x$ in $x$, there exists a packet pair $p_y$ in $y$ such that $p_x$ and $p_y$ occurred consecutively in the same TCP connection. If there are more pairs in $y$ than in $x$, the extra pairs in $y$ are discarded. Finally, this system sorts the sets of packet sequences based on the total number of packets to form a list of packet sequence sets.

*Signature Validation:* The signatures are validated by running the detection algorithm at layer 3 using the dataset on which signatures are created. Suppose the system detects at most $n$ events, and the timestamps of detected events match the timestamps for events recorded during training. In that case, the signature is finalized as a valid packet-level signature.

*Heuristic:* Heuristic is used in both signature generation and detection. At first, we find out the number of packets, $n_i$, of each triggered event within a specific time interval *'t'* (empirically set to 15 milliseconds). Here, $i$ presents different instances of the event and $i = 1, 2, \ldots, m$ as the training dataset generally supports 100 instance per event. Then, we calculate the *geometric mean* considering all the $n_i$ for a given event. If an event is triggered $m$ times during the training phase and the number of packets observed in each instance is $n_1, n_2, \ldots, n_m$, respectively, then the number of packets to be considered will be: $n = \sqrt[m]{\prod_{i=1}^{m} n_i}$. Importantly, we determine the value of $n$ using a controlled setup (lab settings), where no delay exists. For this reason, this is essentially dependent on time. But, since our main goal is to apply our methodology to a real-world scenario, where network traffic contains unavoidable delays, we use there the same $n$ that is not dependent on the time frame anymore, irrespective of datasets and environments. We prove the correctness of our methodology in Section IV-C-2.

We opt for the geometric mean to accommodate unforeseen fluctuations in the traffic rate. If, instead, we use the arithmetic mean and the traffic rate experiences extreme fluctuations, the values of $n_i$ for $m$ number of events will vary significantly, leading to an inappropriate threshold, $n$. Consequently, this will result in inaccurate detection.

❸ *Activity Detection:* Detection is done on independent datasets from which we did not generate signatures. We treat a network trace as a stream of packets. Each packet is presented to a set of state machines. A state machine is maintained for each packet sequence of the signature for each flow, i.e., TCP connection for the WAN sniffer or layer-2 flow for the Wi-Fi

sniffer (both adversaries have been described in Section III-A). A state machine advances to its next state if the packet matches the next packet in the modeled packet sequence. Our heuristic (i.e., Equation III-B) is used here too.

*Signature matching:* There are two signature matching strategies: *exact matching* and *range-based* or *relaxed matching*. In exact matching, only the packets whose lengths exactly match the packet lengths that were observed during training are considered to be valid. On the other hand, in range-based matching, the packet lengths are allowed to lie between a minimum and maximum packet length (i.e., within a small delta, $\epsilon$) observed during training. We set $\epsilon = 10$ as existing work [28]. Now, if the signatures during training are <C-136, S-780> and <C-700, S-511>, the range-based matching will consider client-to-server packets with lengths in $[126, 710]$ and server-to-client packets with lengths in $[501, 790]$ as valid. We used exact matching when no variations in packet lengths were observed during training, otherwise, range-based matching is used. A signature match is searched within $n$ packets (i.e., the heuristic described in Section III-B).

### C. Packet-Based Signature Generation and Detection

We develop a new packet-based signature generation and detection system that relies on the total number of packets to make signatures and detect activities. Algorithm 1 shows the corresponding pseudocode for signature generation and detection of a single event of a given device. At first, we take all packets ($packets$) from the `pcap` file ($F$), events supported by a device ($E$), and the IP address ($IP$) of a device as input. Then, we take a list of timestamps ($TS$) of the occurred event which is stored as separate files. After that, we discard the packets where neither the source nor destination address matches with $IP$ and take the number of packets ($n_i$) occurring within a specific time-window ($t_{window}$) of each time the event is triggered and calculate the heuristic, $n_{heu}$, using equation III-B. Since during training, we determine the value of $n_{heu}$ in controlled lab settings having no delays in traffic, it is dependent on time in essence. However, once we get $n_{heu}$, any real-time inference is not dependent on the time frame anymore. We set $t_{window} = 15$ milliseconds so that we can compare our system with PINGPONG [28]. The detailed procedure is described in Section III-B. Next, we re-iterate the traffic and consider $n_{heu}$ number of packets per event. Finally, we use the DBSCAN clustering algorithm to generate clusters ($clusters$) and concatenate packet pairs in the clusters to reassemble the longest packet sequences possible for creating signatures ($signatures$) of that event. Finally, we output the signatures into a file. Using these signatures and $n_{heu}$, we match signatures and detect activities in any independent dataset, as described in Section III-B.

**Algorithm 1:** Generate Packet-Based Signatures.

**Input**
- **F** Pcap file
- **E** Event of the IoT device
- **IP** IP address of the device
- **TS** Timestamp list of when the event occurred

**Output**
- List of Signatures

$packets \leftarrow$ all packets from $F$
$E \leftarrow$ event name from file
$TS \leftarrow$ list from the timestamps file
$IP \leftarrow$ ip address from command line argument
$n \leftarrow \{\}$
**for** each $pck \in packets$ **do**
  **if** $pck.src \neq IP$ *or* $pck.dest \neq IP$ **then**
  $packets.remove(pck)$
  **end if**
**end for**
$i \leftarrow 0$
**for** each $ts \in TS$ **do**
  Initialize $t_{window}$
  $n[i++] \leftarrow$ number of packets in $[ts, ts + t_{window}]$
**end for**
$n_{heu} \leftarrow Heuristic(m = |TS|, n)$
$considered\_packets \leftarrow []$
**for** each $ts \in TS$ **do**
  $considered\_packets.add(packets(ts, n_{heu}))$
**end for**
$signatures \leftarrow []$
$clusters \leftarrow DBSCAN(considered\_packets)$
**for** each $clstr \in clusters$ **do**
  $sig \leftarrow clstr.packet\_pairs.concatenate()$
  $signatures.add(sig)$
**end for**
**return** $signatures$

### D. Handling Multi-Type Events

For multi-type IoT devices, such as the TP-Link light bulb with events ON-OFF-COLOR-INTENSITY, PINGPONG [28] considers separate events as binary types, and trains and detects them independently. Thus, if there are n possible types of events for a device, PINGPONG runs the detection process at least n/2 times, rather than executing all the events of the device at a time, which results in additional overhead. Consequently, if there are total *x* devices each with *n* possible types of events, then the computational overhead will be $O(xn)$. On the other hand, we improve the system to handle multi-type events simultaneously. Specifically, in multi-type detection, we train and detect all events of a specific device in a single run. As a result, if there are total *x* devices each with *n* possible types of events, then the computational overhead will be only $O(x)$. Therefore, no additional overhead is required, and the performance of the system is improved.

Algorithm 2 shows the corresponding pseudocode for detecting multiple types of events. At first, we merge network traces of all the events of a device into a single `pcap` file. Then, we take all packets ($packets$) from the `pcap` file ($F$), event type list ($Evnt$), and the IP address ($IP$) of the device as input. Next, we take a list of occurred events ($E$) with their corresponding timestamp ($TS$) which were stored in separate files. After that, we discard the packets where neither the source nor destination address

**Algorithm 2:** Handle Multi-type Events.

**Input**
**F** Pcap file
**Evnt** Event type list of IoT device
**IP** IP address of the device
**(E, TS)** Tuple list of type of occurred event with corresponding timestamp
**Output**
List of Signatures
$packets \leftarrow$ all packets from $F$
$Evnt \leftarrow$ event types file
$IP \leftarrow$ ip address from command line argument
$(E, TS) \leftarrow$ events occurred and timestamps file
$n \leftarrow \{\}$
**for** each $pck \in packets$ **do**
  **if** $pck.src \neq IP$ or $pck.dest \neq IP$ **then**
  $packets.remove(pck)$
  **end if**
**end for**
$i \leftarrow 0$
**for** each $t \in Evnt$ **do**
  **for** each $(e, ts) \in (E, TS)$ **do**
    Initialize $t_{window}$
    $n[i++] \leftarrow$ number of packets in $[ts, ts + t_{window}]$
  **end for**
**end for**
$n_{heu} \leftarrow Heuristic(m = |TS|, n)$
$signaturesList \leftarrow []$
**for** each $t \in Evnt$ **do**
  $considered\_packets \leftarrow []$
  **for** each $(e, ts) \in (E, TS)$ **do**
    **if** $e \neq t$ **then**
    ***continue***
    **end if**
    $considered\_packets.add(packets(ts, n_{heu}))$
  **end for**
  $tempList \leftarrow []$
  $clusters \leftarrow DBSCAN(considered\_packets)$
  **for** each $clstr \in clusters$ **do**
    $signature \leftarrow clstr.packet\_pairs.concatenate()$
    $tempList.add(signature)$
  **end for**
  $signaturesList.add(tempList)$
**end for**
**return** $signaturesList$

matches with $IP$ and take the number of packets ($n_i$) within a specific time-window ($t_{window}$) of when an event ($t$) occurs and calculate the heuristic, $n_{heu}$, using III-B. Next, for each type of event ($t$), we iterate over all the times that event occurred $(e, ts)$ and take $n_{heu}$ number of packets. Finally, we use the DBSCAN clustering algorithm to make clusters ($clusters$) and concatenate packet pairs in the clusters to reassemble the longest packet sequences possible for creating signatures ($signatures$) of that event. We output the signatures into a file. For individual events, we keep separate files. Using these signatures and $n_{heu}$, we match signatures and detect activities in other independent datasets, as described in Section IV.

## IV. Evaluation

This section provides a concise overview of our approach. We begin by describing the datasets used for training, testing, and negative, and positive experiments. Next, we present the detailed results of our improved methodologies. We evaluate our system using various performance metrics and compare it with existing work. Additionally, we assess signature uniqueness using network traffic from two large independent datasets and test signature generalizability with another dataset.

### A. Dataset

For evaluation, we used data from four public datasets:

- *Training and Testing: PINGPONG dataset* [28] contains network traces of 19 different smart home devices with a total size of around 40 GB. The dataset was collected in 2018 but it also contains updated traces for some of the devices from 2019. We use this dataset in Section IV-C.
- *Negative Control Experiment:* A negative control experiment means testing our system on an independent dataset to observe the uniqueness of our signatures for uncommon devices.
  - *UNSW smart home traffic dataset* [3] contains network traces for 26 smart home devices. The dataset is a collection of 59 `pcap` files, with a total size of 26.3 GB and a total of 10,497,761 packets. We use this dataset in Section IV-D-1.
  - *YourThings smart home traffic dataset* [26], [27] contains 1,992 `pcap` files from 45 smart home devices, with a total size of approximately 179.9 GB and 282,097,515 packets. We use this dataset in Section IV-D-2.
- *Positive Control Experiment:* A positive control experiment means testing the generalizability of our signatures using an independent dataset with common devices. *Mon(IoT)r* [14] dataset contains network traces for 55 IoT devices, with a total size of around 8.6 GB. We use this dataset in Section IV-E to compare our signatures generated in Section IV-C using the PINGPONG dataset with those generated using this dataset (for only common devices across the datasets).

### B. Examples of Packet-Based Signatures

Fig. 2 illustrates the observed packet exchanges for three IoT devices. For the Sengled bulb, the communication is from the companion app to the cloud. In this case, for an ON event, the controlling smartphone always sends a request packet of 215 bytes to an Internet host and receives a reply packet of 1275 bytes. For the OFF event, these packet lengths are 217 bytes and 1277 bytes, respectively, and for INTENSITY, 211 bytes, and 1063 bytes, respectively. For an Amazon plug, we observe an exchange of TLS Application Data packets between the Amazon plug and an Internet host where the packet lengths are 1099

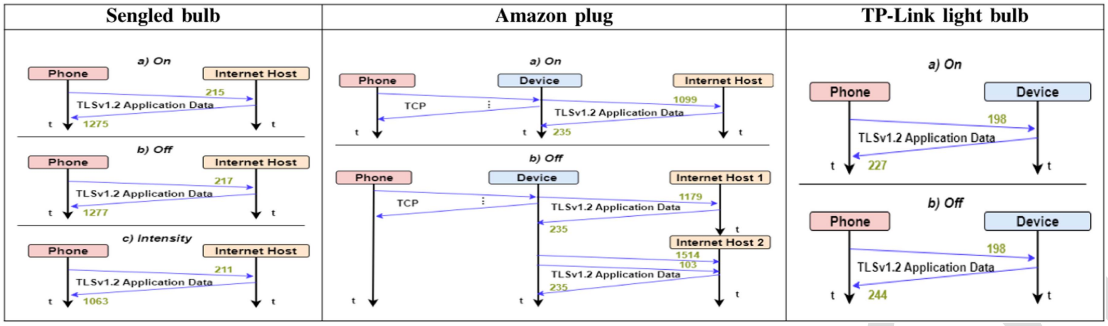


Fig. 2. Packet-level signatures of Sengled bulb, Amazon plug, and TP-Link light bulb observable by a passive network adversary. We can see that unique traffic patterns are exhibited during the execution of specific device events.

TABLE II
RESULTS OF PACKET-BASED SIGNATURE GENERATION & DETECTION METHODOLOGY, INCLUDING DETECTION RESULTS OF PINGPONG

| Device (Communication) | Event | Signature | TPT* | Matching For Ours (Per 100 Events)** | | For PINGPONG | |
|---|---|---|---|---|---|---|---|
| | | | | Validation | Detection | Validation | Detection |
| **Devices with multi-type events** | | | | | | | |
| Sengled bulb (Phone-Cloud) | On | C-215 S-1275 | 11 | 50+50+97=197/200 | 50+49+96=195/200 | 192 | 196 |
| | Off | C-217 S-1277 | | | | | |
| | Intensity | C-211 S-1063 | | | | | |
| TP-Link light bulb (Phone-Device) | On | PH-198 D-227 | 45 | 50+50+100+100=300/300 | 50+50+100+100=300/300 | 300 | 304† |
| | Off | PH-198 D-244 | | | | | |
| | Color | PH-317 D-287 | | | | | |
| | Intensity | PH-[240-242] D-[287-289] | | | | | |
| Rachio sprinkler (Device-Cloud) | Quickrun | S-267 C-155 | 8 | 50+50+100=200/200 | 50+50+100=200/200 | 200 | 200 |
| | Stop | C-496 C-155 | | | | | |
| | Standby | S-299 C-155 C-395 | | | | | |
| | Active | | | | | | |
| **Devices with binary type events** | | | | | | | |
| Ring alarm (Device-Cloud) | Arm | **S1:** S-99 S-254 C-99<br>**S2:** S-[181-183] C-99 | 51 | 49+50=99 | 50+47=97 | 98 | 95 |
| | Disarm | **S1:** S-99 S-255 C-99<br>**S2:** S-[181-183] C-99 | | | | | |
| Ecobee thermostat (Phone-Cloud) | Fan On | C-1387 S-640 | 285 | 50+50=100 | 50+50=100 | 100 | 99 |
| | Fan Off | C-1389 S-640 | | | | | |
| Roomba robot (Phone-Cloud) | Clean | **S1:** S-[1014-1015] C-105<br>**S2:** S-432 C-105 | 87 | 47+47=94 | 52+51=103† | 91 | 94 |
| | Back-to-station | **S1:** S-440 C-105<br>S-[1018-1019, 1023-1024] C-105 | | | | | |
| WeMo plug (Phone-Device) | On | PH-259 PH-475 D-246 | 30 | 100 | 104† | 100 | 100 |
| | Off | | | | | | |
| Dlink plug (Device-Cloud) | On | **S1:** S-91 S-1227 C-784 | 44 | 98 | 96 | 95 | 95 |
| | Off | **S2:** C-1052 S-647 | | | | | |
| Amazon plug (Device-Cloud) | On | C-1099 S-235 | 87 | 49+51=100 | 50+50=100 | 98 | 99 |
| | Off | **S1:** C-1179 S-235<br>**S2:** C-1514 C-103 S-235 | | | | | |
| Kwikset doorlock (Phone-Cloud) | Lock | C-699 S-511 | 38 | 50+50=100 | 54+34=88 | 100 | 100 |
| | Unlock | **S1:** C-701 S-511<br>**S2:** S-647 C-136 | | | | | |
| Dlink siren (Phone-Cloud) | On | C-1076 S-593 | 26 | 50+50=100 | 51+52=103† | 100 | 98 |
| | Off | C-1023 S-613 | | | | | |
| SmartThings plug (Phone-Cloud) | On | C-699 S-511 | 27 | 50+50=100 | 54+41=95 | 92 | 92 |
| | Off | **S1:** C-700 S-511<br>**S2:** S-780 C-136 | | | | | |
| Nest thermostat (Phone-Cloud) | Fan On | C-[891-894] S-[830-834] | 55 | 50+50=100 | 58+59=117† | 93 | 94 |
| | Fan Off | C-[858-860] S-[829-834] | | | | | |
| WeMo Insight plug (Phone-Device) | On | PH-259 PH-475 D-246 | 29 | 100 | 101† | 100 | 99 |
| | Off | | | | | | |
| TP-Link plug (Device-Cloud) | On | C-556 S-1293 | 69 | 50+50=100 | 49+56=105† | 99 | 100 |
| | Off | C-557 S-[1294-1295] | | | | | |

*TPT: Total Packets Taken; **For devices with binary type, 100 events per device. For devices with multi-type events, it varies; † indicates the presence of false positives.
In the signature column, prefixes C, S, PH, and D indicate client-to-server, server-to-client, phone-to-device, and device-to-phone direction, respectively. S1 and S2 are the signature number in the signature column. This means that there are two signatures for that event of the device.

bytes and 235 bytes, respectively, when the plug is ON. For OFF events, we find consistently occurring packet pairs in the plug's communication with two different Internet hosts where the lengths of the reply packets are the same; but for the requests packets, the lengths are different. Similarly, for the TP-Link light bulb, we observe that the smartphone sends a request packet to the device with the same lengths of packets for both events, and the reply packets from the device are of lengths 227 and 244 bytes, respectively. Thus, this request-reply pattern can occur in any communication mode, such as Phone ↔ Device, Device ↔ Cloud, or Phone ↔ Cloud (as highlighted in Section II-B).

### C. Results Using the PINGPONG Dataset

We have tested our methodology on 15 IoT devices on the PINGPONG dataset [28]. Table II depicts the results. For devices with multi-type events, we merge network traces of all the events of a device into a single file and use Algorithm 2 as described in

TABLE III
DIFFERENCE IN GENERATED SIGNATURE BY US AND PINGPONG [28]

| Device* (Communication) | Event | Signature | | Validation | | Detection | |
|---|---|---|---|---|---|---|---|
| | | Ours | PINGPONG | Ours | PINGPONG | Ours | PINGPONG |
| Sengled bulb (Phone-Cloud) | On | C-215 S-1275 | **S1:** C-211 S-1063<br>**S2:** S-1277 | 197 | 192 | 195 | 196 |
| | Off | C-217 S-1277 | C-211 S-1063 S-1276 | | | | |
| | Intensity | C-211 S-1063 | S-[215-217] S-[1275-1277] | | | | |
| Rachio sprinkler (Device-Cloud) | Quickrun | S-267 C-155 | S-267 C-155 | 200 | 200 | 200 | 200 |
| | Stop | C-496 C-155 | C-496 C-155 C-395 | | | | |
| | Standby | S-299 C-155 C-395 | S-299 C-155 C-395 | | | | |
| | Active | | | | | | |
| Roomba robot (Phone-Cloud) | Clean | **S1:** S-[1014-1015] C-105<br>**S2:** S-432 C-105 | **S1:** S-[1014-1015] C-105<br>**S2:** S-432 C-105 | 94 | 91 | 103 | 94 |
| | Back-to-station | **S1:** S-440 C-105<br>S-[1018-1019, 1023-1024] C-105 | **S1:** S-440 C-105<br>S-[1018-1024] C-105 | | | | |
| Amazon plug (Device-Cloud) | On | C-1099 S-235 | **S1:** S-[443-445]<br>**S2:** C-1099 S-235 | 100 | 98 | 100 | 99 |
| | Off | **S1:** C-1179 S-235<br>**S2:** C-1514 C-103 S-235 | **S1:** S-[444-446]<br>**S2:** C-1179 S-235<br>**S3:** C-1514 C-103 S-235 | | | | |
| SmartThings plug (Phone-Cloud) | On | C-699 S-511 | **S1:** C-699 S-511<br>**S2:** S-777 C-136 | 100 | 92 | 95 | 92 |
| | Off | **S1:** C-700 S-511<br>**S2:** S-780 C-136 | **S1:** C-700 S-511<br>**S2:** S-780 C-136 | | | | |

*Devices, only for which we get new signatures, are stated here.
We get some new signatures that are smaller in size (i.e., vary in packet length) or completely different from the signatures reported by PINGPONG. The highlighted portions of the table indicate the major differences.

Section III-D. For most of the devices, we can detect signatures for different events. The fourth column shows the number of packets we have used for signature generation and detection. The fifth column of the table contains the validation results conducted on the training dataset, and the sixth column contains the detection results after testing on traffic, independent of the training and validation phases. Finally, the last two columns contain corresponding validation and detection results for PINGPONG. The "TPT (Total Packets Taken)" depends on individual network traffic, that is the total number of packets in the `pcap` file, which varies from device to device. Though we discard irrelevant packets, we can not strictly say the "majority" of the packets are deemed irrelevant and subsequently discarded. In several cases, we find most of the packets to be irrelevant. On the contrary, in other cases, the network traffic contains comparatively fewer packets. Interestingly, we get identical signatures for the individual events of the WeMo plug, Dlink plug, and WeMo Insight Plug (possibly these devices share the same software codebase from the same vendor). Also, for some device events we obtain different signatures as highlighted using $S1$ and $S2$ labels in the table. In the "Matching" results of the last two columns, we denote the number of detected events of each type of device by summation. For example, our system detects 50 On, 50 Off, and 97 Intensity events of the Sengled bulb device in the validation phase. However, the total number of actual events for the Sengled bulb is 200. Therefore, we mention the results as "50+50+97=197/200" in the table. For devices with binary events, since the total number of actual events is 100 per device in both validation and detection, we skip mentioning the number of actual events in the table. Similar things go for the detection column and other devices, except for the three devices (WeMo plug, Dlink plug, and WeMo Insight Plug), for which we get identical signatures, we state the number of detected events together, rather than writing separately.

We next compare our signatures with the ones reported by PINGPONG [28]. We discover some *new signatures* for five devices: Sengled bulb (ON-OFF-INTENSITY), Rachio sprinkler (STOP), Roomba robot (BACK-TO-STATION), Amazon plug (ON-OFF), and SmartThings plug (ON). These new signatures are either smaller in size (i.e., vary in packet length, deduct any part of the signature) or completely different from the signatures stated in PINGPONG [28]. Table III illustrates a comparison between our system and PINGPONG [28], only for the devices where we obtain new signatures. For the Sengled bulb, we get signatures <C-215 S-1275>, <C-217 S-1277>, and <C-211 S-1063> for the ON, OFF, and INTENSITY events, respectively, which are completely different from those reported by PINGPONG [28]. These new signatures improve the validation rate while keeping the detection rates almost identical. For Rachio sprinkler's event STOP, the signature generated by PINGPONG [28] is <C-496 C-155 C-395>. In our experiment, $C-395$ is not present, and the new signature becomes <C-496 C-155>. Importantly, this shortened signature does not affect the validation and detection system, as evident in Table III. The signature in PINGPONG [28] for the event BACK-TO-STATION for the Roomba robot is <S-440 C-105 S-[1018-1024] C-105>. On the contrary, our system generates a smaller signature excluding $S-[1020-1022]$, and again we observe improvement in both the validation and detection phases. For the Amazon plug, parts of PINGPONG's [28] signatures: $S-[443-445]$ and $S-[444-446]$, for events ON and OFF, respectively, are not present in our new signatures (see in Table III). Most importantly, these new signatures prove their correctness by obtaining perfect matches in the validation and detection phases (i.e., 100 versus 98, 100 versus 99). Finally, for the SmartThings plug's ON event, our system comes up with a single signature, that is, <C-699 S-511>. On the other hand, the corresponding existing signatures from PINGPONG [28] are <C-699 S-511> and <S-777 C-136> (i.e., two signatures). From Table III, we can see that this new signature can identify more events compared to PINGPONG [28] (i.e., 100 versus 92 and 95 versus 92).

*1) Comparisons With PINGPONG:* We have compared our system with the PINGPONG [28] system in terms of signature

TABLE IV
COMPARISON BETWEEN BOTH SYSTEMS ON SIGNATURE DURATION

| Device | Duration Avg. (Max., Min.) (ms) | |
|---|---|---|
| | Ours | PINGPONG |
| Sengled bulb | 5311 (7171, 3321) | 5964.5 (8151.5, 4145.5) |
| TP-Link bulb | 83 (212, 6) | 83.3 (212, 6) |
| Rachio sprinkler | 1412 (2538, 276) | 1435 (2538, 276) |
| Ring alarm | 410 (605, 275) | 410 (605, 275) |
| Ecobee thermostat | 232 (1776, 117) | 232 (1776, 117) |
| Roomba robot | 2057 (5418, 123) | 2038 (5418, 123) |
| Wemo plug | 42 (134, 33) | 42 (134, 33) |
| Dlink plug | 1206 (8060, 823) | 1206 (8060, 823) |
| Amazon plug | 686 (1466, 70) | 2465 (4537, 1232) |
| Kwikset doorlock | 305 (2361, 136) | 395 (2874, 173) |
| Dlink siren | 37 (65, 36) | 37 (65, 36) |
| SmartThings plug | 385 (2223, 131) | 537 (2223, 335) |
| Nest thermostat | 111 (1072, 91) | 111 (1072, 91) |
| Wemo Insight plug | 39 (97, 32) | 39 (97, 32) |
| TP-Link plug | 85 (204, 75) | 85 (204, 75) |

duration and detection. The average signature duration for our approach across all the devices is 1,141 milliseconds, which is less than that of PINGPONG (1,499 milliseconds). From Table IV, we can see that the longest signature in PINGPONG is 9,132 milliseconds (for Sengled bulb), whereas in our case, it is 8,060 milliseconds (for Dlink plug). Moreover, our system improves the minimum and average signature duration time.

For the Sengled bulb, the minimum duration reduces to 3,324 milliseconds from 4,145.5 milliseconds, the average duration reduces to 5,311 milliseconds from 5,964.5 milliseconds, and the maximum duration reduces to 7,171 milliseconds from 9,132 milliseconds. For the Amazon plug, we observe a drastic change in signature duration time as minimum, average, and maximum duration reduced by 1,162 milliseconds (70 versus 1,232 ms), 1,179 milliseconds (686 versus 2,465 ms), and 3,071 milliseconds (1,466 versus 4,537 ms), respectively. For the Kwikset doorlock, the minimum duration reduces to 136 milliseconds from 173 milliseconds, the average duration reduces to 305 milliseconds from 395 milliseconds, and the maximum duration reduces to 2,361 milliseconds from 2,874 milliseconds. Finally, for the SmartThings plug, minimum and average signature duration improve (131 versus 335 ms, and 385 versus 537 ms), and maximum duration remains similar to PINGPONG. For four devices: Sengled bulb, Amazon plug, Kwikset doorlock, and SmartThings plug, our performance is much better than PINGPONG. For others, it remains almost the same.

Moreover, the last four columns of Table II illustrate that our system always gives better validation results since it can, most of the time, validate all 100 triggered events. In the case of detection, it either improves or remains consistent with the performance of PINGPONG. In our experiments, a true positive is a case where our system can detect the event that is actually triggered, a false positive is where there is no event happening but our system detects as that event occurred, and a false negative is a case where our system can not detect the actual triggered event. However, a true negative is not possible here because there will be nothing to detect if there is no ongoing event. In any detection system, the target is to get high true positives and low false negatives. From Table V, we can see that our system gives more true positives and fewer false negatives than the existing

TABLE V
COMPARISON BETWEEN OUR SYSTEM AND PINGPONG

| Metrics | Procedures | Ours | PINGPONG |
|---|---|---|---|
| Total number of True Positives | Validation | 1888 | 1858 |
| | Detection | 1871 | 1861 |
| Total number of False Negatives | Validation | 12 | 42 |
| | Detection | 29 | 39 |
| Recall | Validation | 99.3% | 97.4% |
| | Detection | 98.5% | 97.4% |
| Precision | Validation | 100% | 100% |
| | Detection | 98.3% | 99% |

We achieve higher recall and similar precision rates.

system since our main goal is to reduce false negatives and increase true positives, which may result in a few false positive events as a side effect. Besides, we get a higher recall score which indicates that we can detect more events than others. Additionally, the precision of both systems is almost similar. Therefore, the results of Tables V and IV indicate that our system *outperforms* PINGPONG in the signature generation, and detection process.

*Takeaway.* PINGPONG [28] employs a time window-based signature generation and detection strategy, considering all packets within a specific time-window. However, this approach's drawback lies in its sensitivity to traffic rate fluctuations, leading to variations in validation and detection phases. This is especially problematic during network congestion, as fewer packets may be available for signature detection in such periods. To address these limitations, we have introduced a packet-count-based approach (Section III-B), which mitigates the impact of temporal fluctuations and network congestion. Moreover, this approach remains unaffected by higher-than-usual traffic rates, ensuring accurate signature detection.

*2) Experiment on Real World Settings:* In the real world, network traffic usually experiences random delays up to 500 ms, even 700 ms [16], [33]. Therefore, to better justify our methodology and prove our claim, we create a dataset by injecting random delays in the PINGPONG dataset [28] and test both systems on the new dataset. Specifically, we inject delays of different ranges, for example [0-100] ms, [100-200] ms, [200-300] ms, [300-400] ms, and [400-500] ms varying the frequency of the delays, such as after every 2, 5, 7, 10, 25, 50, 75, and 100 packets. For every experiment, at first, we select a delay range, let [0-100] ms. Then, we randomly choose a delay from that range every time, keeping the frequency of the delays static for that experiment. After that, we repeat the same experiment 10 times (rounds), and finally, we take the average of the detection results. Similarly, we conduct experiments changing the frequency for that delay range. As a result, in total, there are 5*7*10=350 experiments per device. Fig. 3 shows the comparison results on precision, recall, and f1-score metrics. We see that PINGPONG predicts few true positive events, but misses many of them (resulting in too many false negative events). Thus, in most cases, there are near 0 false positives. Consequently, recall and accuracy of PINGPONG drastically decrease to 14.7% and 25.6%, respectively, as the more frequent delays increase. Additionally, precision becomes almost 100 since there are a negligible number of false positives. On the other hand, the precision, recall, and f1-score

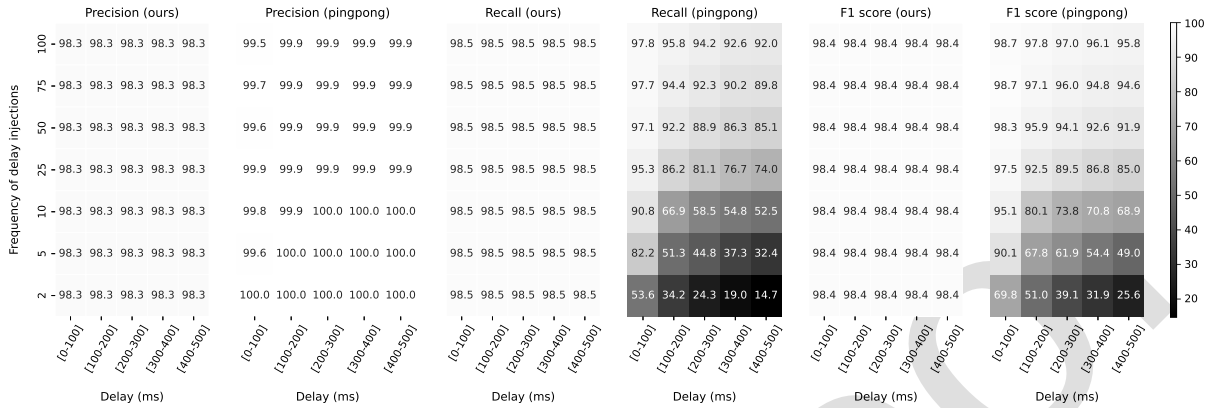


Fig. 3. In real-world comparisons, our system maintains stable performance metrics. In contrast, PINGPONG experiences a notable decrease in recall and F1-score, while precision remains unaffected. This suggests that while PINGPONG accurately detects true events, it misses a significant number of them.

of our system remain stable at 98.3%, 98.5%, and 98.4%, respectively, which means the system is not affected by the delays at all.

### D. Negative Control Experiments

*1) Using UNSW Dataset:* UNSW dataset [3] contains network traces of 26 IoT devices. We evaluate the uniqueness of the signatures generated for 15 IoT devices by performing signature detection on the UNSW dataset [3]. As there is *no common devices* between UNSW and the 15 IoT devices for which we have generated signatures, all the detected events will be considered false positives. We get a total *90* false positive events across a total of *10,497,761* packets.

*2) Using YourThings Dataset:* YourThings dataset [27] contains network traces of 45 IoT devices. We have tested our system on the YourThings dataset, which includes 1,992 `pcap` files over seven days. As there are three common devices (WeMo plug, Roomba robot, and TP-Link light bulb) present in both the YourThings dataset and our training-testing dataset, we only performed signature detection for 12 of our devices that are not present in the YourThings dataset to avoid the potential for true positives. We get here only *258* false positives across a total of *282,097,515* packets.

*Takeaway.* In the two negative control experiments, we observe an average of *1* false positive per *0.117 million* packets in the UNSW dataset and an average of *1* false positive per *1.09 million* packets in the YourThings dataset. These false positives occur due to the *range-based or relaxed* matching strategy employed by our system. This strategy considers a match even if the packet lengths in independent traffic are close to the exact lengths observed during training. The detailed strategy is explained in Section III-B.

### E. Positive Control Experiment Using Mon(IoT)r Dataset

In this Section, we apply our methodology to the publicly available Mon(IoT)r [14] dataset. We compare the signatures extracted from the Mon(IoT)r dataset to those extracted from the PINGPONG [28] dataset for the devices present in both datasets. There are five common devices: WeMo Insight plug, Blink camera, TP-Link plug, Sengled bulb, and TP-Link light bulb. Since Mon(IoT)r dataset was collected in 2019, we use the secondary dataset that PINGPONG collected for the common devices in 2019. Thus, we repeat the signature extraction for the dataset collected in 2019 for these devices to facilitate a better comparison of signatures.

Table VI highlights the signatures for the common devices across the two independent datasets. We can see that the signatures for the three devices changed. For the WeMo Insight plug and Blink camera, the extracted signatures from the PINGPONG (2019) dataset and the Mon(IoT)r dataset are identical. The signatures for the TP-Link plug and Sengled bulb remain almost the same across these two datasets. The packet lengths slightly differ in some positions by a few bytes. For example, in the PINGPONG dataset, the TP-Link plug's OFF signature is <C-593 S-[1235-1236] S-100>, which changes to <C-606 S-[1214-1215] S-100> in Mon(IoT)r dataset. A similar thing goes for the Sengled bulb. We could not obtain an updated network trace from the PINGPONG dataset for the TP-Link light bulb. Thus, we used the previous version. Overall, signatures across these two datasets do not exactly match, but they are still closer.

Another interesting insight is the temporal stability of signatures. Comparing Tables II with VI, we can see that the signature changed for two of the devices across a 2-year time span. For example, the WeMo Insight plug's signature no longer contains $PH - 259$, and $S - 100$ is added to the TP-Link plug's signature.

*Takeaway.* We observed in the positive control experiment that some signatures change over time, probably due to the configuration changes, communication protocol introduced in firmware updates, or other credentials. This evolution can result in introducing false negative events as the signatures will not match the ones that were observed during training. To overcome this situation, we need to update our signatures periodically. The best thing is to repeat the training process to extract the latest signatures of a device right before launching the detection process.

TABLE VI
COMMON DEVICES IN THE MON(IOT)R AND PINGPONG DATASETS

| Device | Event | Signature | TPT † | Duration (ms) Min./Avg./Max./St. Dev. |
|---|---|---|---|---|
| WeMo Insight plug (Phone-Device) | On/Off | *S1: PH-475 D-246 | 139 | 29 / 33 / 112 / 8.51 |
| | | **S1: PH-475 D-246 | 313 | 31 / 42 / 111 / 14.97 |
| Blink camera (Device-Cloud) | Watch | *S1: C-331 S-229 C-139 | 635 | 267 / 273 / 331 / 8.35 |
| | | **S1: C-331 S-229 C-139 | 71 | 170 / 269 / 289 / 17.57 |
| | Photo | *S1: C-331 C-123 S-139 S-123 S-187 C-1467 | 96 | 281 / 645 / 1299 / 345 |
| | | **S1: C-331 C-123 S-139 S-123 S-187 C-1467 | 44 | 266 / 1307 / 11146 / 2232.5 |
| TP-Link plug (Device-Cloud) | On | *S1: C-592 S-[1234-1235] S-100 | 61 | 70 / 73 / 85 / 2.2 |
| | Off | *S1: C-593 S-[1235-1236] S-100 | | |
| | On | **S1: C-605S-[1213-1214] S-100 | 69 | 10 / 77 / 189 / 40.85 |
| | Off | **S1: C-606 S-[1214-1215] S-100 | | |
| Sengled bulb (Device-Cloud) | On/Off | *S1: S-[216-218] C-[208-210] *S2: C-430 | 10 | 3251 / 4657 / 6118 / 862.57 |
| | On | **S1: S-219 C-210 **S2: C-428 **S3: C-[478-479] | 17 | 354 / 2592 / 3836 / 861.39 |
| | Off | **S1: S-219 C-210 **S2: C-428 **S3: C-[479-480] | | |
| TP-Link light bulb (Phone-Device) | On | *S1: PH-198 D-227 | 88 | 8 / 77 / 148 / 42.2 |
| | Off | *S1: PH-198 D-244 | | |
| | On | **S1: PH-258 D-227 | 79 | 3905 / 6685 / 8071 / 389.7 |
| | Off | **S1: PH-258 D-244 | | |

*Signature: Training on PINGPONG [28]; **Signature: Training on Mon(IoT)r [14]; †TPT: Total Packets Taken.
We see minimal changes across the datasets, if any.

## V. DISCUSSION

Our system demonstrates high accuracy in revealing IoT device activities from real smart home network traffic logs. The experimental evaluations, covering a diverse range of IoT devices, show an average recall of 98-99% and an average precision of 98-100%. We identify new packet-level signatures and validate their uniqueness by performing detection on independent datasets, resulting in negligible false positives. Furthermore, the generated signatures exhibit good generalizability within the same time frame, although firmware updates can impact the final signature.

Our system has a few limitations. First, the proposed methodology can only be applied to the TCP protocol, not to the UDP protocol, which we plan to explore in the future. Besides, we observe a few false positives as a side effect of reducing false negatives while maximizing true positives.

Strategies to prevent general traffic analysis, like website fingerprinting defenses [20], can mitigate our signature detection system. These include using VPNs, traffic padding, and shaping techniques, which aim to obscure traffic patterns and connection details. However, they often introduce significant bandwidth and latency overheads, challenging the real-time nature of IoT devices. Other approaches involve injecting adversarial noise into traffic [29] or splitting traffic across multiple networks [8], albeit with practical limitations. A promising approach akin to *'k-anonymity'* could group traffic based on network usage by IoT devices while preserving usability. Implementation and evaluation of such methods for IoT devices remain avenues for future research.

## VI. CONCLUSION

This paper introduces a packet-based signature generation and detection system for effectively detecting user activities of diverse IoT devices, encompassing single, binary, and multi-type events, using packet-level signatures extracted from both encrypted and unencrypted network traffic. By overcoming the time-window dependency limitations of existing systems, we demonstrate the system's superiority and correctness through experimental results in real-world datasets. The uniqueness and accuracy of our generated signatures are validated using publicly available datasets, yielding satisfactory outcomes.

## REFERENCES


[1] A. Acar et al., "Peek-a-boo: I see your smart home activities, even encrypted!," in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Linz, Austria, 2020, pp. 207–218.

[2] A. Sivanathan et al., "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2017, pp. 559–564.

[3] A. Sivanathan et al., "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, Aug. 2019.

[4] D. Ahmed, A. Das, and F. Zaffar, "Analyzing the feasibility and generalizability of fingerprinting Internet of Things devices," *Proc. Privacy Enhancing Technol.*, vol. 2022, no. 2, pp. 578–600, Mar. 2022.

[5] A. Panchenko et al., "Website fingerprinting at internet scale," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016. page= None Q3

[6] B. Copos, K. N. Levitt, M. Bishop, and J. Rowe, "Is anybody home? Inferring activity from smart home network traffic," in *Proc. IEEE Secur. Privacy Workshops*, 2016, pp. 245–251.

[7] DataFlair Team, "How IoT works-4 main components of IoT system," [Online]. Available: https://data-flair.training/blogs/how-iot-works/ year= 2019 Q4

[8] W. De la Cadena et al., "TrafficSliver: Fighting website fingerprinting attacks with traffic splitting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1971–1985.

[9] G. Acar et al., "FPDetective: Dusting the web for fingerprinters," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 1129–1140.

[10] I. Hafeez, A. Y. Ding, M. Antikainen, and S. Tarkoma, "Real-time IoT device activity detection in edge networks," in *Proc. Int. Conf. Netw. Syst. Secur.*, 2018, pp. 221–236.

[11] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "IoT-KEEPER: Detecting malicious IoT network activity using online traffic analysis at the edge," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 45–59, Mar. 2020.

[12] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proc. USENIX Secur. Symp.*, 2015, pp. 1187–1203.
[13] Y. Jin, E. Sharafuddin, and Z.-L. Zhang, "Unveiling core network-wide communication patterns through application traffic activity graph decomposition," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 1, pp. 49–60, 2009.
[14] J. Ren et al., "Information exposure for consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. Internet Meas. Conf.*, 2019, pp. 267–279.
[15] J. Ren et al., "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. Internet Meas. Conf.*, 2019. pages={267--279},
[16] J. Li, T. Zhang, J. Jin, Y. Yang, D. Yuan, and L. Gao, "Latency estimation for fog-based Internet of Things," in *Proc. 27th Int. Telecommun. Netw. Appl. Conf.*, 2017, pp. 1–6.
[17] M. Lopez-Martin, B. Carro, A. J. Sánchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.
[18] M. Miettinen et al., "IoT sentinel: Automated device-type identification for security enforcement in IoT," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 2177–2184.
[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231.
[20] N. Mathews, J. K. Holland, S. E. Oh, M. S. Rahman, N. Hopper, and M. Wright, "SoK: A critical evaluation of efficient website fingerprinting defenses," in *Proc. IEEE Symp. Secur. Privacy*, 2023, pp. 344–361.
[21] T. T. T. Nguyen and G. J. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surveys Tut.*, vol. 10, no. 4, pp. 56–76, Fourth Quarter 2008.
[22] N. J. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart(ER) IoT traffic shaping," *Proc. Privacy Enhancing Technol.*, vol. 2019, pp. 128–148, 2018.
[23] N. J. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic," 2017, *arXiv: 1705.06805*.
[24] N. J. Apthorpe, D. Reisman, and N. Feamster, "Closing the blinds: Four strategies for protecting smart home privacy from network observers," 2017, *arXiv: 1705.06809*.
[25] N. J. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic," 2017, *arXiv: 1708.05044*.
[26] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Yourthings scorecard," [Online]. Available: https://yourthings.info/ year= 2019
[27] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security evaluation of home-based IoT deployments," in *Proc. IEEE Symp. Secur. Privacy*, San Francisco, CA, USA, 2019, pp. 1362–1380.
[28] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home devices," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2020. page= None
[29] M. S. Rahman, M. Imani, N. Mathews, and M. Wright, "Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1594–1609, 2021.
[30] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "IoTFinder: Efficient large-scale identification of IoT devices via passive DNS traffic analysis," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2020, pp. 474–489.
[31] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2010, pp. 391–404.
[32] R. Doshi, N. J. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in *Proc. IEEE Secur. Privacy Workshops*, 2018, pp. 29–35.
[33] A. S. S. M. D. Kumar, "Delay estimation of healthcare applications based on MQTT protocol: A node-red implementation," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol.*, 2022, pp. 1–6.
[34] S. Marchal, M. Miettinen, T.D. Nguyen, A.-R. Sadeghi, and N. Asokan, "AuDI: Toward autonomous IoT device-type identification using periodic communication," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1402–1412, Jun. 2019.
[35] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," *J. Comput. Commun.*, vol. 3, pp. 164–173, Dec. 2022.
[36] A. Subahi and G. Theodorakopoulos, "Detecting IoT user behavior and sensitive information in encrypted IoT-app traffic," *Sensors*, vol. 19, no. 21, 2019, Art. no. 4777.
[37] T. OConnor et al., "HomeSnitch: Behavior transparency and control for smart home IoT devices," in *Proc. 12th Conf. Secur. Privacy Wireless Mobile Netw.*, Miami, FL, USA, 2019, pp. 128–138.
[38] Z. Berkay Celik et al., "Sensitive information tracking in commodity IoT," in *Proc. 27th USENIX Conf. Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1687–1704.

**Mohammad Shamim Ahsan** received the BS degree in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET), in 2023. Currently, he is a lecturer with the Computer Science and Engineering (CSE) Department, United International University, Dhaka, Bangladesh. He will start his PhD in Fall'24 with the IST Department of Pennsylvania State University, USA. His research interests are in Cybersecurity and Privacy, focusing on IoT security, computer security, web security, and social aspects of security. Apart from this, he is continuously serving society as a peer-reviewer in several well-known journals.

**Md. Shariful Islam** received the BSc degree in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2023. Currently, he works as a software engineer with Enosis Solutions, Bangladesh, while seeking PhD opportunities starting Fall '25. He possesses experience in IoT security and has participated in Computer Security competitions like Capture the Flag. His research interests encompass IoT security, cryptography, web security, and network security.

**Md. Shohrab Hossain** (Member, IEEE) received the BSc and MSc degrees in computer science and engineering from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2003 and 2007, respectively, and the PhD degree from the School of Computer Science, University of Oklahoma, Norman, OK, USA, in 2012. During his PhD, he worked under NASA-funded projects related to survivability, scalability, and security of space networks. He is currently serving as a professor with the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. His research interests include Cyber Security, Mobile malware detections, Software-defined networking (SDN), security of mobile and ad hoc networks, and the Internet of Things. He has published more than 90 technical research papers in leading journals and conferences including the *Journal of Computers & Security*, *Ad Hoc Networks*, *IEEE Access*, *Journal of Network and Computer Applications*, *IEEE Trans. of Mobile Computing*, *Wireless Personal Communication*, *PLoS One*, IEEE GLOBECOM, IEEE ICC, IEEE MILCOM, IEEE WCNC, IEEE HPCC, etc.

**Anupam Das** (Senior Member, IEEE) received the BS and MS degrees in computer science and engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2008 and 2010, respectively, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign (UIUC), Illinois, USA, in 2016. He was a recipient of a Fulbright Science and Technology fellowship during his PhD degree. He was also a postdoctoral fellow with the School of Computer Science, Carnegie Mellon University (CMU) from 2016 to 2018. He is currently an assistant professor with the Computer Science Department at North Carolina State University (NCSU), Raleigh, North Carolina, USA. His research interests lie in security and privacy, with a special focus on designing secure and privacy-preserving technologies. He has published more than 60 technical research papers in leading journals and conferences. He has been awarded more than U.S. $2.5 million in research grants from the U.S. National Science Foundation (NSF), NCSU, Meta, and Amazon. He has also received two ACM Distinguished Paper Awards (ASIACCS 2014, MMSys 2017). His projects have been covered by media outlets such as Wired, Forbes, ZDNet, MotherBoard, and FastCompany.