Inferring User Activities from IoT Device Events in Smart Homes: Challenges and Opportunities

Xuanli Lin, Yinxin Wan, Kuai Xu, Feng Wang, Guoliang Xue

***Abstract*—The ubiquitous deployment of IoT devices in smart homes has led to growing research interests in studying the home network traffic for various applications such as network measurements, device profiling, and IoT device event inference. Recent studies have shown that user activities can be inferred from a home network using extracted device event logs. However, existing solutions for user activity inference such as `IoTMosaic` and `E2AP` have limitations when handling ambiguities caused by device malfunctions. In this paper, we first identify the challenges faced by the existing user activity inference algorithms and the root causes of their poor performances on certain types of inputs. We then show that useful information can still be obtained even in situations where device malfunctions introduce ambiguities in user activity patterns. We achieve so by designing an extension to the existing algorithms. We also apply our extension in a digital forensics application. Our extensive experimental evaluations demonstrate that our solutions can effectively provide insights to user activity inference despite the presence of indistinguishable user activity patterns.**

***Index Terms*—Internet-of-things, device events, user activity inference, challenges and opportunities.**

1. Introduction

The rapid growth and wide deployment of the Internet-of-Things (IoT) have sparked their applications in many emerging and crucial fields such as industry automation, smart city, and connected health. In particular, a variety of smart home IoT devices are available on the market which bring advances in home security, remote controlling, and intelligent device management. The availability of heterogeneous IoT devices in smart home environments has driven research in network traffic measurement [15, 20, 23], device profiling and classification [7, 8, 11, 12, 17, 25], and device events extraction [1, 19, 21, 26].

Some recent studies have shown that it is possible to infer user activities in a smart home where different kinds of IoT devices are deployed by monitoring the home network traffic [1, 22, 24]. The extraction of user activities in [22, 24] relies on accurate identification of IoT device events from the home network [19, 21] and is based on the assumption that a user activity always generates a distinct and ordered sequence of device events. However, missing and out-of-order device events due to potential device malfunctions or network instabilities bring substantial challenges to existing solutions.

To address the above issues, [22] proposes an approximate signature matching algorithm that allows some of the device events in a user activity's signature to be missing when matching a user activity to its signature in the observed device event sequences. On the other hand, [24] considers different patterns of a user activity where some device events could be missing, and proposes an unsupervised machine learning technique for learning the optimal weights of all activity patterns that result in the most accurate user activity inference.

In this paper, we study user activity inference in smart homes by examining the challenges to existing solutions. In particular, we present several impossibility results. We show by examples that missing events (caused by device malfunction, for example) could make different user activities indistinguishable in the observed device events. Not surprisingly, the accuracy of the existing inference algorithms such as `IoTMosaic` [22] and `E2AP` [24] decreases when there are activities that share the same device event sequence after device malfunctions.

We show that even in the presence of ambiguous user activity patterns, we can still obtain useful information by designing an extension to the existing solutions. The key to our approach is to identify the patterns that could be generated by the user activities, and report all possible user activities that share the same pattern. We demonstrate that this approach can lead to effective solutions to an application in digital forensics.

The main contributions of this paper are the following:

- We perform a detailed analysis on challenges that affect the accuracy of the existing activity inference algorithms such as `IoTMosaic` [22] and the `E2AP` scheme [24].
- We propose an extension to the `E2AP` framework to address some of the challenges.
- We present an application of the proposed changes in a digital forensics setting to show that our approach can provide useful information even in the presence of indistinguishable user activities.

The remainder of this paper is organized as follows. Section 2 defines the problem of user activity inference in smart homes and provides a high level overview of the `E2AP` algorithm in [24]. Section 3 discusses the challenges to existing approaches in user activity inference and presents some impossibility results. Section 4 explores the opportunities in improving the performance of existing algorithms for user activity inference. Performance evaluation results are presented in Section 5. Section 6 discusses related works. Section 7 concludes the paper and outlines future research works.

2. Problem Formulation and the E2AP Solution

In this section, we present the *Device Events to User Activity Patterns* (`E2AP`) problem studied in [24] and outline the

All authors are affiliated with Arizona State University. Emails: {xlin54, ywan28, xlin54, kuai.xu, fwang25, xue}@asu.edu. This research was supported in part by NSF grants 2007469, 2007083, 1816995, and 1717197. The information reported here does not reflect the position or the policy of the funding agency.

essential parts of the solution framework proposed in [24]. The materials presented here are needed in Section 3 where we analyze the challenges to existing solutions, as well as in later sections where we explore opportunities to improve upon the existing solutions.

A. The E2AP Problem Formulation

As in [24], $\mathcal{A} = \{A^1, A^2, \ldots, A^{|\mathcal{A}|}\}$ is used to denote a set of *user activities*, where each user activity $A \in \mathcal{A}$ can trigger a sequence of IoT *device events* $\mathbb{S}^1(A) = (e_1^A, e_2^A, ..., e_{|\mathbb{S}^1(A)|}^A)$ or a subsequence of $\mathbb{S}^1(A)$ with some missing device events. Each of such sequence of device events that can be triggered by user activity A is called a *possible pattern* of A. Let $\mathcal{S}(A) = \{\mathbb{S}^1(A), \mathbb{S}^2(A), \ldots, \mathbb{S}^{|\mathcal{S}(A)|}(A)\}$ be the set of all possible patterns of user activity A, where $\{\mathbb{S}^2(A), \ldots, \mathbb{S}^{|\mathcal{S}(A)|}(A)\}$ are proper subsequences of $\mathbb{S}^1(A)$.

Given a sequence of device events $\mathbb{E} = (e_1, e_2, \ldots, e_m)$, the E2AP problem aims to find a sequence of user activities $\mathbb{A} = (A_1, A_2, \ldots, A_{|\mathbb{A}|})$ together with their patterns $\mathbb{S}_j^{k_j} \in \mathcal{S}(A_j)$ for $j = 1, 2, \ldots, |\mathbb{A}|$, such that $\mathbb{S}_1^{k_1} \| \mathbb{S}_2^{k_2} \| \cdots \| \mathbb{S}_{|\mathbb{A}|}^{k_{|\mathbb{A}|}}$ is as close to $\mathbb{E}$ as possible. Here $\|$ denotes the concatenation binary operator.

B. The E2AP Solution Framework

To facilitate the understanding of this paper, we briefly describe the basic concepts and the main ideas of the solution framework proposed in [24] for solving the E2AP problem.

A *match* of pattern $\mathbb{S}^k(A) = (e_1^{A,k}, e_2^{A,k}, \ldots, e_{|\mathbb{S}^k(A)|}^{A,k})$ in $\mathbb{E}$ is denoted by $\psi_i^{A,k}, i \in \mathbb{Z}^+$, which is a strictly increasing sequence of positive integers $(\psi_{i,1}^{A,k}, \psi_{i,2}^{A,k}, \ldots, \psi_{i,|\mathbb{S}^k(A)|}^{A,k})$ such that $e_{\psi_{i,j}^{A,k}} = e_j^{A,k}, \forall 1 \le j \le |\mathbb{S}^k(A)|$. The *interval* of $\psi_i^{A,k}$ is $\psi_i^{A,k}.interval = [\psi_{i,1}^{A,k}, \psi_{i,|\mathbb{S}^k(A)|}^{A,k}]$. A match $\psi_i^{A,k}$ is *minimal* if there does not exist another match $\psi_{i'}^{A,k}$ whose interval is a proper subset of the interval of $\psi_i^{A,k}$. A match $\psi_i^{A,k}$ is *lighter* than another match $\psi_{i'}^{A,k}$ if $\psi_{i,|\mathbb{S}^k(A)|}^{A,k} < \psi_{i',|\mathbb{S}^k(A)|}^{A,k}$, or $\psi_{i,|\mathbb{S}^k(A)|}^{A,k} = \psi_{i',|\mathbb{S}^k(A)|}^{A,k}$ and $(\psi_{i,|\mathbb{S}^k(A)|-1}^{A,k}, \psi_{i,|\mathbb{S}^k(A)|-2}^{A,k}, \ldots, \psi_{i,1}^{A,k})$ is lexicographically greater than or equal to $(\psi_{i',|\mathbb{S}^k(A)|-1}^{A,k}, \psi_{i',|\mathbb{S}^k(A)|-2}^{A,k}, \ldots, \psi_{i',1}^{A,k})$. If a match $\psi_i^{A,k}$ is lighter than another match $\psi_{i'}^{A,k}$, then $\psi_{i'}^{A,k}$ is *heavier* than $\psi_i^{A,k}$. Two matches $\psi_i^{A,k}$ and $\psi_{i'}^{A,k}$ are *equivalent* if their intervals are the same. All matches that are equivalent to each other form an equivalent class. We use the lightest match in each equivalent class as its *representative*. A two-phase scheme is then designed in [24] for solving E2AP. We use the following two examples to illustrate the main ideas of phase I and phase II of the scheme in [24], respectively.

Example 1: Let $\mathbb{E} = (a, b, c, a, b, a, b, d, c)$, $\mathcal{A} = (A^1, A^2)$, $\mathcal{S}(A^1) = \{(a, b, c), (a, b)\}$, and $\mathcal{S}(A^2) = \{(d, c)\}$. The ground truth user activity sequence $\mathbb{A}$ is (A^1, A^1, A^1, A^2). In phase I, E2AP applies a matching algorithm named AkMatch to compute a sequence of representatives of the equivalence classes for any given $A \in \mathcal{A}$, and any $k \in \{0, 1, \ldots, |\mathcal{S}(A)|$. There are 7 matches of $\mathbb{S}^1(A^1) = (a, b, c)$ in $\mathbb{E}$ whose indices are $(1, 2, 3), (1, 2, 9), (1, 5, 9), (1, 7, 9), (4, 5, 9), (4, 7, 9), (6, 7, 9)$, and they form 4 equivalent classes: $\{$**(1, 2, 3)**$\}$, $\{(1, 2, 9)$, $(1, 5, 9), ($**1, 7, 9**$)\}$, $\{(4, 5, 9)$, **(4, 7, 9)**$\}$, $\{$**(6, 7, 9)**$\}$, where the representatives are marked in bold. Out of these 4 equivalent classes, 2 representative minimal matches $\{$**(1, 2, 3)**$\}, \{$**(6, 7, 9)**$\}$ will be found by AkMatch. Similarly, AkMatch outputs 3 representative minimal matches **(1, 2)**, **(4, 5)**, **(6, 7)** for $\mathbb{S}^2(A^1)$ and 1 representative minimal match **(8, 9)** for $\mathbb{S}^1(A^2)$. □

Phase I only needs to be executed once which allows us to focus on a much smaller subset of all matches without losing critical information. However, the matches of two different patterns identified by AkMatch may be mutually exclusive with each other, thus resulting in collisions. An example is the matches **(1, 2, 3)** of pattern $\mathbb{S}^1(A^1)$ and **(1, 2)** of pattern $\mathbb{S}^2(A^1)$ in Example 1. To output the sequence of matches that would most likely resemble the actual activity sequence while preventing conflicts among them, [24] associates a weight to each pattern such that a match of the pattern with higher weight has higher priority. Given the assigned weights for all patterns, phase II of the framework in E2AP adopts the OMatch algorithm to find a sequence of compatible (not mutually exclusive) matches that have the maximum total weight.

In OMatch, a match is further characterized by a 5-tuple (i, k, α, β, w) where i is the index of the user activity A^i, k is the index of the pattern $\mathbb{S}^k(A^i)$ of A^i, α and β are the starting and ending index of the match in $\mathbb{E}$, and w is the weight $w(i, k)$ assigned for $\mathbb{S}^k(A^i)$. Given a sequence of 5-tuples $\mathbb{M}$, OMatch algorithm finds a subset $\mathbb{M}_w \in \mathbb{M}$ that does not conflict with each other and has the maximum total weights. Here OMatch is essentially the dynamic programming algorithm for maximum weighted interval scheduling. The following example illustrates how OMatch works. Note the settings in Example 2 are the same as Example 1.

Example 2: First, the representative matches in Example 1 are sorted according to the β field which result in $\mathbb{M}[1] = (1, 2, 1, 2, w(1, 2))$, $\mathbb{M}[2] = (1, 1, 1, 3, w(1, 1))$, $\mathbb{M}[3] = (1, 2, 4, 5, w(1, 2))$, $\mathbb{M}[4] = (1, 2, 6, 7, w(1, 2))$, $\mathbb{M}[5] = (2, 1, 8, 9, w(2, 1))$, and $\mathbb{M}[6] = (1, 1, 6, 9, w(1, 1))$. Assume the weights are given as $w(1, 1) = 2.0, w(1, 2) = 0.5, w(2, 1) = 0.5$. OMatch starts with $\mathbb{M}[1]$, which is picked because picking $\mathbb{M}[1]$ results in a larger weight than not picking it, for it is the first match in $\mathbb{M}$. $\mathbb{M}[2]$ is examined next, which would conflict with $\mathbb{M}[1]$. Because $\mathbb{M}[2].w = w(1, 1) > w(1, 2) = \mathbb{M}[1].w$, $\mathbb{M}[1]$ is replaced by $\mathbb{M}[2]$. $\mathbb{M}[3]$, $\mathbb{M}[4]$, and $\mathbb{M}[5]$ are added because there are no match conflicts with them yet. However, $\mathbb{M}[6]$ is picked by OMatch to replace both $\mathbb{M}[4]$ and $\mathbb{M}[5]$ because $\mathbb{M}[4]$ and $\mathbb{M}[5]$ conflict with $\mathbb{M}[6]$ and $\mathbb{M}[6].w = w(1, 1) > w(1, 2) + w(2, 1) = \mathbb{M}[4].w + \mathbb{M}[5].w$. OMatch thus outputs the matches $\mathbb{M}_w = (\mathbb{M}[2], \mathbb{M}[3], \mathbb{M}[6])$ which correspond to user activities (A^1, A^1, A^1) and result in the maximum total weight without any conflicts.

However, if the weight assignment is $w(1, 1) = 1.0, w(1, 2) = 0.7, w(2, 1) = 0.7$, OMatch will output $\mathbb{M}_w = (\mathbb{M}[1], \mathbb{M}[3], \mathbb{M}[4], \mathbb{M}[5])$ which correspond to user

activities (A^1, A^1, A^1, A^2). □

Example 2 shows that different weight assignments could result in different sequences of user activities. To measure the quality of the matches, [24] calculates the Levenshtein edit distance between the device event sequence $\mathbb{E}$ and the device event sequence assembled by concatenating the patterns of the matched user activity sequence generated by `OMatch`.

For example, the device event sequence assembled by the matches $(\mathbb{M}[2], \mathbb{M}[3], \mathbb{M}[6])$ in Example 2 is (a, b, c, a, b, a, b, c), whose edit distance from $\mathbb{E}$ is 1. On the other hand, the the device event sequence assembled by the matches $(\mathbb{M}[1], \mathbb{M}[3], \mathbb{M}[4], \mathbb{M}[5])$ in Example 2 is $(a, b, c, a, b, a, b, d, c)$ which has an edit distance of 0 from $\mathbb{E}$.

Thus it is crucial to find a set of weights so that `OMatch` will output the matches whose concatenated patterns are as close to $\mathbb{E}$ as possible. To address this problem, [24] designs an unsupervised learning approach to learn a proper weight assignment with the loss function set as the edit distance between the assembled device event sequence and $\mathbb{E}$. A coordinated descent approach is adopted to optimize each of the weights individually in a round-robin fashion until the loss function stops decreasing. Detailed descriptions of the algorithms can be found in [24].

3. Understanding the Challenges

While both [22] and [24] show great promises in inferring user activities from a sequence of IoT device events (obtained from the network traffic using existing techniques such as [21]), the accuracies of both approaches drop when the missing device events result in different user activities correspond to identical sequences of device events. This could happen, for example, when there are device malfunctions.

The primary objectives of this paper are to understand the challenges in inferring user activities in the face of many missing device events, and to explore the opportunities where we can address some of the challenges. We examine the challenges in this section, and explore the opportunities in Section 4. As we will see in this section, some of the challenges are intrinsic to the problem itself. Therefore, instead of trying to design algorithms for obtaining impossible results, we should set more realistic goals and try to design algorithms to obtain the best possible results.

A. Identical Patterns

Let $\mathbb{S}^1(A^1) = (a, b, c)$ and $\mathbb{S}^1(A^2) = (a, b)$. Suppose that event c can be missing due to intermittent device malfunctions. As such, A^1 can trigger the sequence $\mathbb{S}^2(A^1) = (a, b)$, which is a subsequence of $\mathbb{S}^1(A^1)$. Note that the sequences $\mathbb{S}^2(A^1)$ and $\mathbb{S}^1(A^2)$ are identical. Hence, if we observe the pattern (a, b), we cannot tell whether it is triggered by user activity A^1 or by user activity A^2.

Suppose that the observed sequence of device events is $\mathbb{E} = (a, b, a, b, a, b, c)$. Since we cannot reliably distinguish between $\mathbb{S}^2(A^1)$ and $\mathbb{S}^1(A^2)$, we can find four distinct sequences of user activity patterns that all resemble the input $\mathbb{E}$, i.e., $(\mathbb{S}^1(A^1), \mathbb{S}^1(A^2), \mathbb{S}^1(A^2))$, $(\mathbb{S}^1(A^1), \mathbb{S}^1(A^2), \mathbb{S}^2(A^1))$, $(\mathbb{S}^1(A^1), \mathbb{S}^2(A^1), \mathbb{S}^2(A^1))$, and $(\mathbb{S}^1(A^1), \mathbb{S}^2(A^1), \mathbb{S}^1(A^2))$. These four sequences of activity patterns correspond to the sequences of user activities (A^1, A^2, A^2), (A^1, A^2, A^1), (A^1, A^1, A^1), and (A^1, A^1, A^2), respectively.

Theorem 1: If two or more user activities share an identical pattern $\mathbb{P}$, then it is impossible to accurately infer (from the sequence of device events) which user activity triggered a sequence of device events that is identical to $\mathbb{P}$. □

While the impossibility result in Theorem 1 seems to imply that inferring user activities from device events is extremely challenging, we can set more realistic goals in designing algorithms to infer user activities. For a pattern $\mathbb{P}$ that is a possible pattern for both user activity A^1 and user activity A^2, we can claim with high confidence that $\mathbb{P}$ *is triggered by either* A^1 *or* A^2. As we will see in the next section, such information will be very useful in certain situations.

B. Ambiguous Combinations

Let $\mathbb{S}^1(A^2) = (a, b)$, $\mathbb{S}^1(A^3) = (c, d)$, and $\mathbb{S}^1(A^4) = (a, b, c, d)$. It is apparent that $\mathbb{S}^1(A^4)$ can be formed by concatenating $\mathbb{S}^1(A^2)$ and $\mathbb{S}^1(A^3)$.

Given the sequence of device events $\mathbb{E} = (a, b, c, d)$, two possible sequences of user activity patterns can be inferred: $(\mathbb{S}^1(A^2), \mathbb{S}^1(A^3))$ or $(\mathbb{S}^1(A^4))$. The `OMatch` algorithm would choose the pattern(s) whose weight is the highest, i.e. choose $(\mathbb{S}^1(A^2), \mathbb{S}^1(A^3))$ if $w(2,1) + w(3,1) > w(4,1)$, and choose $(\mathbb{S}^1(A^4))$ if $w(2,1) + w(3,1) < w(4,1)$. As a consequence, despite having the same resulting device events, the algorithm may overlook occurrences of activities that are deemed less important. Hence, we have the following impossibility result.

Theorem 2: If the concatenation of a sequence of user activity patterns is identical to the concatenation of a different sequence of user activity patterns (use $\mathbb{P}$ to denote this concatenation), then it is impossible to accurately infer (from the sequence of device events) which sequence of user activities triggered a sequence of device events that is identical to $\mathbb{P}$. □

As we discussed at the end of Section 3-A, this impossibility result does not stop us from exploring opportunities to infer useful information. We will discuss these opportunities in Section 4.

C. Empty Subsequences

Let $\mathbb{S}^1(A^2) = (a, b)$ and $\mathbb{S}^1(A^5) = (b)$. Suppose event b can be missing, then user activity A^2 also has a possible pattern $\mathbb{S}^2(A^2) = (a)$. Similarly, user activity A^5 also has a possible pattern $\mathbb{S}^2(A^5) = ()$. Note that $\mathbb{S}^2(A^5)$ is an empty sequence.

Since $\mathbb{S}^2(A^5)$ is an empty sequence, a user activity inference algorithm (such as the ones in [22] and [24]) can possibly infer an arbitrary number of occurrences of user activity A^5. Therefore we have the following impossibility result.

Theorem 3: If a user activity A has a possible pattern that is the empty sequence, then it is impossible to accurately infer (from the sequence of device events) the occurrence(s) of user activity A. □

The situation discussed in Theorem 3 may happen due to device malfunctions. While this is unavoidable, we can deal with this situation using a simple technical convention. For possible patterns that are empty sequences, we simply ignore all of them. As a result, the corresponding user activity/activities will not appear in our inferred result. While not ideal, this is the best solution we can come up with.

4. Exploring Opportunities

In the previous section, we discussed several challenging types of input for the E2AP framework. They prevent the algorithms from accurately inferring user activities from IoT device events. In this section, we show that useful information can still be obtained to answer certain queries regarding user activities, even in scenarios where different user activities have identical patterns due to device malfunctions.

A. Getting More Out of the Computed Activity Patterns

As explained in Section 3-A and Theorem 1, when two or more activities correspond to the same sequence of device events, it becomes impossible to reliably distinguish between these activities using only the input device event sequence. As a consequence, the algorithms in the E2AP present only the activity with the highest weight, ignoring other possible matches.

We can, however, use the proposed solution $\mathbb{S}^w$ from the E2AP and perform an interpretation on top of it, where different possibilities are accounted for. Here $\mathbb{S}^w = (\mathbb{S}^w[1], \mathbb{S}^w[2], \ldots, \mathbb{S}^w[|\mathbb{M}_w|])$, where $\mathbb{S}^w[j] = \mathbb{S}^{\mathbb{M}_w[j].k}(A^{\mathbb{M}_w[j].i})$, $j = 1, 2, \ldots, |\mathbb{M}_w|$. Recall that $\mathbb{M}_w$ is the output of OMatch. The approach is simple - we scan the sequence $\mathbb{S}^w$ in order, and for each match $\mathbb{S}^w[j]$ in $\mathbb{S}^w$, we check if there are any activity patterns that have the same sequence as $\mathbb{S}^w[j]$, and provide these patterns as alternatives to the original solution along with $\mathbb{S}^w[j]$.

After this operation, we obtain a list $\mathbb{L}_j$ with all activity patterns that have the same event sequence as $\mathbb{S}^w[j]$. This list can contain only $\mathbb{S}^w[j]$ if no other activity patterns have the same event sequence as $\mathbb{S}^w[j]$. Then we add each $\mathbb{L}_j$ to a new sequence $\overline{\mathbb{S}}^w$, and return the sequence $\overline{\mathbb{S}}^w$ when we finish scanning $\mathbb{S}^w$.

We call this interpretation E2AP+, and the approach is illustrated in the example below.

Example 3: Consider the following input: $\mathbb{E} = (a, b, c, d, a, b, c, d, e, a, b, c)$; $\mathcal{A} = \{A^1, A^2, A^3\}$; $\mathbb{S}^1(A^1) = (a, b, c, d), \mathbb{S}^2(A^1) = (a, b, c)$; $\mathbb{S}^1(A^2) = (a, b, c)$; $\mathbb{S}^1(A^3) = (d, e)$.

Using E2AP, we obtain the following activity sequence: $\mathbb{S}^w = (\mathbb{S}^1(A^1), \mathbb{S}^2(A^1), \mathbb{S}^1(A^3), \mathbb{S}^2(A^1))$. That is, the algorithm reports that activities (A^1, A^1, A^3, A^1) have occurred.

Since $\mathbb{S}^2(A^1) = \mathbb{S}^1(A^2)$, it is possible that we come across cases where $\mathbb{S}^1(A^2)$ instead of $\mathbb{S}^2(A^1)$ has occurred. Because they correspond to the same pattern, we can provide $\mathbb{S}^1(A^2)$ as an alternative to $\mathbb{S}^2(A^1)$ so both subsequences can be considered.

Using this approach, we can expand the activity sequence to the following: $\overline{\mathbb{S}}^w = (\mathbb{S}^1(A^1), [\mathbb{S}^2(A^1)$ or $\mathbb{S}^1(A^2)], \mathbb{S}^1(A^3), [\mathbb{S}^2(A^1)$ or $\mathbb{S}^1(A^2)])$. This means one of the following activity sequences has occurred: (A^1, A^1, A^3, A^1), (A^1, A^1, A^3, A^2), (A^1, A^2, A^3, A^1), or (A^1, A^2, A^3, A^2). □

To evaluate the performance of E2AP+, we propose a modified edit distance algorithm that can work with the sequence $\overline{\mathbb{S}}^w$ where alternatives are provided with the original activity subsequences, and it is listed in Algorithm 1.

Algorithm 1: CalcED($\overline{\mathbb{S}}^w, \mathbb{A}$)

Input: $\overline{\mathbb{S}}^w = (\mathbb{L}_1, \mathbb{L}_2, \ldots, \mathbb{L}_{|\mathbb{M}_w|})$: computed activity sequence from E2AP+;
$\mathbb{A} = A_1, A_2, \ldots, A_n$: ground truth sequence of user activities;
Output: edit distance between the two sequences

```
/* perform initialization on c                          */
1  for i := 0 to n do
2      c[i, 0] ← i;
3  for j := 0 to |M_w| do
4      c[0, j] ← j;
5  for i := 1 to n do
6      for j := 1 to |M_w| do
7          if Equivalent(L_j, A_i) then
               /* L_j and A_i are equivalent, no
                  edit distance is added                 */
8              c[i, j] ← c[i − 1, j − 1];
9          else
               /* L_j and A_i are not equivalent,
                  add 1 to edit distance                 */
10             c[i, j] ← min(c[i − 1, j − 1],
                             c[i, j − 1],
                             c[i − 1, j]) + 1;
11 output c[n, |M_w|].
```

The behavior of the Equivalent function on line 7 depends on the operation mode of CalcED. Here, we specify two modes of operations for CalcED: strict and relaxed. In the strict mode, we count $\mathbb{L}_j$ and A_i being equivalent if $\mathbb{L}_j$ only contains one activity pattern $\mathbb{S}^k(A^{i'})$, and $A^{i'} = A_i, k \in \{1, \ldots, |\mathcal{S}(A^{i'})|\}$. In the relaxed mode, we count $\mathbb{L}_j$ and A_i being equivalent as long as there exists an activity pattern $\mathbb{S}^k(A^{i'})$ in $\mathbb{L}_j$, where $A^{i'} = A_i$ and $k \in \{1, \ldots, |\mathcal{S}(A^{i'})|\}$.

As such, given the same input sequence $\overline{\mathbb{S}}^w$, the reported performance from CalcED can differ drastically depending on the operation mode. We detail the comparison of the two modes in Section 5.

The E2AP+ we proposed in this section can be further developed to solve problems that would be difficult otherwise with the original E2AP framework. We show one of its usage in a digital forensics setting in the following.

B. Application in Digital Forensics

Consider a smart home where its owners are interested in knowing whether someone entered their home while they are away. Using the smart home setup in [22], we focus on

Activity 1 to Activity 6 in Table II of [22] which correspond to a person entering the home through the front door [without key/using an app/using a key] at [day/night]. These user activities are closely related to home security. Thus, we can infer whether and when someone has entered the home by looking for occurrences of Activity 1 to Activity 6 in the device event sequence.

Given the requirements of this problem, the output of E2AP may be unsatisfactory. As we explained in Example 3, when multiple activity patterns share the same device event sequence, E2AP finds only one of the possible activity patterns. However, in the digital forensics scenario, the owner might be interested in all possible occurrences of a user activity, and the E2AP approach results in the loss of information. Therefore, we present two approaches that alleviate this issue in the following sections.

The `FindAct` Approach

The first approach is a match finding method that incorporates the E2AP+ discussed in the previous section. Note that in this scenario, we are effectively using the relaxed metric because it is better to have false positives than to miss actual activity occurrences. This approach is presented in Algorithm 2.

Algorithm 2: FindAct($\mathbb{S}^w, \mathcal{S}, A$)

Input: $\mathbb{S}^w = (\mathbb{S}^w[1], \mathbb{S}^w[2], \ldots, \mathbb{S}^w[|\mathbb{M}^w|])$: sequence of matched activities from E2AP with their timestamps $t_1, t_2, \ldots, t_{|\mathbb{M}^w|}$;
$\mathcal{S}$: set containing all activity patterns;
A: the activity we are interested in;
Output: $\mathbb{T}$: list containing timestamps of all possible occurrences of A in $\mathbb{S}^w$

```
𝕊̄^w ← ();
for j := 1 to |𝕄^w| do
    𝕃_j ← (𝕊^w[j]);
    /* Find all activity patterns that are
       equal to 𝕊^w[j]                          */
    for i := 1 to |𝒮| and k := 1 to |𝒮(A^i)| do
        if 𝕊^w[j] = 𝕊^k(A^i) then
            𝕃_j.append(𝕊^k(A^i));
    𝕊̄^w.append(𝕃_j);
/* Find all occurrences of A in the
   expanded activity pattern sequence           */
𝕋 ← ();
for j := 1 to |𝕄^w| do
    foreach 𝕊^k'(A^i') in 𝕊̄^w[j] do
        if A^i' = A then
            𝕋.append(t_j);
output 𝕋.
```

The algorithm works as follows. We first use E2AP+ on a sequence of activities $\mathbb{S}^w$ found by the E2AP to construct a sequence $\overline{\mathbb{S}}^w$ of activity patterns with their alternatives. The new sequence, along with timestamps for each activity, can be used to determine the times when a particular activity may have happened. If we are interested in when activity A took place, we scan $\overline{\mathbb{S}}^w$ and check whether activity A possibly occurred at time t_j in $\overline{\mathbb{S}}^w[j]$ (either as a single activity or among the list of possible activities). If we find activity A in $\overline{\mathbb{S}}^w[j]$, we add the corresponding timestamp t_x to the list of results. The resulting list $\mathbb{T}$ thus contains all possible occurrences of activity A given $\mathbb{S}^w$.

To illustrate Algorithm 2 as well as the subsequent algorithm in this section, we will use the following setting where $\mathbb{E}$ is illustrated as:

index	1	2	3	4	5	6	7	8	9	10	11
event	b	c	a	b	a	b	c	d	c	b	c
time	1	5	6	7	12	14	15	18	22	25	28

Note the notion of time here: in [24], the time attributes are present in $\mathbb{E}$ but they were unused, as the main focus was to infer a chronologically correct sequence of activities without considering actual timestamps of the activities. Here, we define the timestamp of a match of an activity to be the timestamp of the first device event in that match.

Further, let $\mathcal{A} = \{A^1, A^2\}$ where $\mathcal{S}(A^1) = \{(a, b, c), (b, c)\}$ and $\mathcal{S}(A^2) = \{(b, c)\}$. The ground truth activity sequence with timestamps is $\mathbb{A} = ((A^2, 1), (A^1, 6), (A^1, 12), (A^2, 25))$.

Example 4: Suppose we are interested in occurrences of activity 2. Using the settings laid out above, we first run E2AP and the algorithm outputs $\mathbb{S}^w = ((\mathbb{S}^2(A^1), 1), (\mathbb{S}^1(A^1), 12), (\mathbb{S}^2(A^1), 25))$.

We scan $\mathbb{S}^w$ and find that $\mathbb{S}^2(A^1) = \mathbb{S}^1(A^2)$, thus we add $\mathbb{S}^1(A^2)$ as an alternative to $\mathbb{S}^2(A^1)$. There are no alternatives to $(\mathbb{S}^1(A^1)$. Thus, after running E2AP+, we find the following sequence of activities $\overline{\mathbb{S}}^w = (([\mathbb{S}^2(A^1), \mathbb{S}^1(A^1)], 1), ([\mathbb{S}^1(A^1)], 12), ([\mathbb{S}^2(A^1), \mathbb{S}^1(A^1)], 25))$.

We then conclude that activity 2 may have taken place at time 1 and 25, which fully matches the ground truth. Note that activity 2 was not originally in $\mathbb{S}^w$, but we were able to uncover it with FindAct. □

This example shows that Algorithm 2 can find some occurrences of a user activity that would otherwise be missing in the original E2AP. Algorithm 2 however does not fundamentally change the match finding strategy in E2AP, but only expands on its results. As such, if we miss an activity completely, FindAct would be unable to recover relevant information. For example, if we are instead interested in finding activity 1, then either way we can only get the occurrence at time 12, but not the occurrence at time 6.

This observation prompts the need for an alternative match finding strategy where all occurrences of the user activity we are interested in are considered.

The `WMatch` Approach

The second approach attempts to find the timestamp of all possible occurrences of user activity A using a sequence of device events $\mathbb{E}$. However, we cannot accept all matches because the occurrences of activity A are bounded in a time frame. For example, if the average duration for "a person entering the home through the front door" is 10 seconds, it would not make sense to include matches that are 5 minutes

long. Therefore, a duration threshold is necessary to indicate a duration above which the activity is unlikely to occur.

Taking this consideration into account, we add the time duration constraint to the problem of detecting occurrences of activities in a given device event sequence. Here an occurrence of an activity is defined by its start time t_α. Let $A \in \mathcal{A}$ be the activity we are interested in. This activity can correspond to one of its possible patterns in $\mathcal{S}(A)$. Let $\mathbb{E}$ be a sequence of device events (with their timestamps), and let θ be a positive real number that represents the maximum duration threshold (we do not include A in the results if it spans longer than this duration).

A subsequence $\mathbb{E}'$ in $\mathbb{E}$ (with start time t_α and end time t_β) is a time-constrained match to $\mathbb{S}^k(A)$ if both of the following criteria are met:

- $\mathbb{E}'$ is a match to $\mathbb{S}^k(A)$ (according to the definitions in Section 2);
- $t_\beta - t_\alpha \leq \theta$.

To find all time-constrained matches for the activity of interest using the device event sequence $\mathbb{E}$, we design the `WMatch` algorithm, listed in Algorithm 3. The algorithm, though based on `AkMatch` from the `E2AP`, has two distinctions from it.

First, instead of scanning the whole input event sequence for matches at once, we scan only a small subsequence at a time. Specifically, from a starting index x, we scan events from e_x up to the event with the timestamp no larger than $t_x + \theta$. This enforces the duration threshold we impose on the matches.

Second, we search for the heaviest time-constrained matches rather than the lightest matches in these subsequences.

Finding the heaviest time-constrained matches means that we would find the longest possible duration (under the time threshold) for the activities we are interested in. This ensures we do not miss out possible time period when the activity would be still ongoing.

Example 5: Using the example settings in Section 4-B, we perform matching for $S^1(A^1) = (a, b, c)$. The following matches can be found: $(3, 4, 7), (3, 4, 9), (3, 4, 11), (3, 6, 7), (3, 6, 9), (3, 6, 11), (3, 10, 11), (5, 6, 7), (5, 6, 9), (5, 6, 11), (5, 10, 11)$.

In heaviest first order, we have $(3, 4, 11), (3, 6, 11), (3, 10, 11), (5, 6, 11), (5, 10, 11), (3, 4, 9), (3, 6, 9), (5, 6, 9), (3, 4, 7), (3, 6, 7), (5, 6, 7)$. □

We first explain the data structures used in Algorithm 3.

- c is an integer matrix of $m + 1$ rows by $\eta + 1$ columns. Each filled entry corresponds to $|\texttt{LCS}(\mathbb{E}(i_\mu : i), \mathbb{S}^k(A)(1 : j))|$.
- i_μ is the start index (on $\mathbb{E}$) of the current search.
- i_ν is the last index (on $\mathbb{E}$) where $\mathbb{S}^k(A)[\eta]$ is found.
- $\mathbb{D}$ is a sequence that stores time-constrained matches found by the algorithm. `append` operation adds a new match to the sequence at the end.
- l is an integer that denotes the index number of the next match.
- h_l is a sequence containing the l-th match. `prepend` operation adds a new element to the front of the sequence.

Algorithm 3: WMatch($\mathbb{E}, A, k, \theta$)

Input: $\mathbb{E} = (e_1, e_2, ..., e_m)$: input sequence of device events with their timestamps $t_1, t_2, ...t_m$;
A: activity of interest;
k: pattern index of activity A;
θ: duration threshold for A;

Output: $\mathbb{D}$: a sequence of matches found, sorted by their order of occurrence;

```
/* perform initialization                                   */
1  η ← |S^k(A)|; i ← 1; l ← 0; D ← ();
2  i_μ ← 1;
3  while i_μ ≤ m do
       /* search for the next suitable
          starting point                                    */
4      while i_μ + 1 ≤ m and e_{i_μ} ≠ S^k(A)[1] do
5          i_μ ← i_μ + 1;
6      i ← i_μ; i_ν ← 0;
       /* assign 0 to all entries in c                     */
7      c[i, j] ← 0, 1 ≤ i ≤ m, 1 ≤ j ≤ η;
       /* iterate through events till we hit
          max duration allowed                              */
8      while i ≤ m and t_i ≤ t_{i_μ} + θ do
9          for j := 1 to η do
10             if e_i = S^k(A)[j] then
11                 c[i, j] ← c[i − 1, j − 1] + 1;
12                 if j = η then
13                     i_ν ← i;
14             else
15                 c[i, j] ← max(c[i, j − 1], c[i − 1, j]);
16         i ← i + 1;
       /* check if a match is found                        */
17     if c[i − 1, η] = η then
18         l ← l + 1; h_l ← (); row ← i − 1; col ← η;
           /* backtrack to reconstruct the
              match                                         */
19         while col > 0 do
20             if col = η then
21                 h_l.prepend(i_ν);
22                 col ← col − 1; row ← row − 1;
23             else
24                 while c[row − 1, col] = c[row, col] do
25                     row ← row − 1;
26                 h_l.prepend(row);
27                 col ← col − 1; row ← row − 1;
28         D.append(h_l);
29     i_μ ← i_μ + 1;
30 output D.
```

The flow of Algorithm 3 is the following. Lines 1, 2, 6, 7 perform initialization on the data structures. The algorithm then enters a while loop (lines 3-29) which operates on a floating start index i_μ. In lines 4 and 5, we find the next suitable starting index which has $e_{i_\mu} = \mathbb{S}^k(A)[1]$, the first event in $S^k(A)$. This ensures we do not waste time on start indexes that would not yield correct results. Once such starting index is found, we proceed to perform matching till we either hit the end of the event sequence $\mathbb{E}$ or the match found would

exceed the maximum allowed duration θ. In lines 8 to 16, we dynamically find the length of LCS between $\mathbb{E}(i_\mu : i)$ and $\mathbb{S}^k(A)(1 : j)$. This ensures that we find the heaviest time-constrained matches of $\mathbb{S}^k(A)$ in the current search area. Note that in lines 12 and 13, we keep track of i_ν, which is the most recently discovered index where $e_{i_\nu} = \mathbb{S}^k(A)[\eta]$. This is done to facilitate backtracking once we find a match.

Then, in line 17 we check whether a match is found. If $|\texttt{LCS}(\mathbb{E}(i_\mu : i), \mathbb{S}^k(A))|$ is equal to the length of $|\mathbb{S}^k(A)|$, a match is found for $\mathbb{S}^k(A)$ and we proceed to backtrack and reconstruct the match in lines 18 to 28. First, we set the end of the match to i_ν, which ensures the match we reconstruct is the heaviest. We then proceed to backtrack the rest of the match till we find all events in $\mathbb{S}^k(A)$. The new match is added to the list of matches in line 28, and we move the start index i_μ forward by one in line 29. This guarantees the next match found is different from the previous one.

Example 6: Using the same example setting in Section 4-B. Suppose we are interested in activity A^1, with its device event pattern $\mathbb{S}^1(A^1) = (a, b, c)$. Additionally, we allow for a duration threshold θ of 10. The following table illustrates the algorithm using this example. Note that the first column and the first row correspond to event indices in $\mathbb{E}$ and $\mathbb{S}^1(A^1)$, respectively. The second column shows each event in $\mathbb{E}$ with its timestamp in the (event, timestamp) format, while the second row shows the events in $\mathbb{S}^1(A^1)$. The rest of the table shows the length of LCS of the two sequences: $|\texttt{LCS}(\mathbb{E}(i_\mu : i), \mathbb{S}^1(A^1)(1 : j))|$ for each coordinate (i, j).

TABLE I: Running examples of `WMatch` with $i_\mu = 3$ and $i_\mu = 5$.

		1	2	3
		a	b	c
1	(b, 1)	0	0	0
2	(c, 5)	0	0	0
3	(a, 6)	1	1	1
4	(b, 7)	1	2	2
5	(a, 12)	1	2	2
6	(b, 14)	1	2	2
7	(c, 15)	1	2	3
8	(d, 18)	0	0	0
			...	

(a) $i_\mu = 3$

		1	2	3
		a	b	c
			...	
4	(b, 7)	0	0	0
5	(a, 12)	1	1	1
6	(b, 14)	1	2	2
7	(c, 15)	1	2	3
8	(d, 18)	1	2	3
9	(c, 22)	1	2	3
10	(b, 25)	0	0	0
11	(c, 28)	0	0	0

(b) $i_\mu = 5$

We start with $i_\mu = 1$. Since $e_1 \neq \mathbb{S}^1(A^1)[1]$, we skip forward to the next starting index. Similarly $e_2 \neq \mathbb{S}^1(A^1)[1]$, and we skip index 2 as well. On $i_\mu = 3$, we find that $e_3 = \mathbb{S}^1(A^1)[1]$. We proceed to look for the heaviest match from the 3rd event up till the last event e' with $t_{e'} \leq t_3 + \theta$. In this case it is event 7 at timestamp 15 ($t_7 = 15 < 6 + 10$). As shown in Table Ia, we have shaded the area from index 3 to 7 where the search takes place. We then begin to perform matching. At the end of the current search area, we have $c[7, 3] = 3 = |\mathbb{S}^1(A^1)|$. This indicates a time-constrained match of $\mathbb{S}^1(A^1)$ is found. Note that this iteration terminated because we would be running over the duration threshold if we continue to event 8, not because we found a match.

Since a match is found, we backtrack to reconstruct the heaviest match. We start with assigning the saved i_ν index to $h_l[3]$. Recall that i_ν corresponds to the index on $\mathbb{E}$ where $S^k(A)[\eta]$ last occurred. In this case it is $S^1(A^1)[3] = c$ that last occurred on index 7. We then find the elements in the heaviest match, resulting in b on index 4, and a on index 3. The match is thus $(3, 4, 7)$ that starts at time 6 and ends at time 15.

Likewise, we can find another time-constrained match $(5, 6, 9)$ that starts at time 12 and ends at time 22, as illustrated in Table Ib. We continue to increment i_μ and eventually we reach the end of the input sequence $\mathbb{E}$. The algorithm terminates with two time-constrained matches: $\{(3, 4, 7), (5, 6, 9)\}$. □

As illustrated in Example 6, we are able to find both occurrences of activity 1 using `WMatch`. We further evaluate the performance of both Algorithms 2 and 3 in the following section.

5. Experimental Evaluations

In this section, we present the evaluation results for our proposed solutions. We first introduce our experiment settings, and then we present the results of the performance evaluation on the `E2AP+` in two modes of `CalcED`. We also compare the performance of `FindAct`, `WMatch`, and `IoTMosaic` from [22] in the digital forensics setting in Section 4.

A. Experiment Setting

To evaluate the performance of the algorithms, we made use of the activity and event sequences from [22] that were obtained in a real-life smart home test bed. The sequences are divided into three lengths, 387, 1494, and 2959 activities, corresponding to the first week, first month, and full two months of the experiment. We designate these the `real` data, in which there are 21 different user activities involving 11 unique IoT devices and 25 device events. Detailed descriptions for these activities, devices, and events can be found in [22].

We analyzed the `real` data, and generated a synthetic dataset with the characteristics of the `real` data. Specifically, the distributions of the activities are roughly the same in the synthetic data and the `real` data. The duration for each occurrence of an activity A^i is given by a random variable Θ_i whose density is given by a normal distribution $N \sim (\mu_i, \sigma_i^2)$. The parameters of this distribution are calculated using the samples of A^i in the `real` data. These procedures ensure the test synthetic data retains most characteristics of the real-world data while introducing some randomness. The synthetically generated data has lengths of 387, 1494, 2959, and 10000 to better illustrate and compare the results with [24] and [22].

For each of the four different lengths of user activity sequence, we generated two categories of test cases with different difficulty levels: (I) `easy`, where there are no ambiguities in activity mappings, which is the same as the `synth` dataset in [24]; (II) `hard`, where around 29% of user activities have patterns that share the same sequence of device events. We refer to these dataset as the `synth-new` data.

B. Performance Evaluation of E2AP+

First, we evaluate the performance of the E2AP+ using the synth-new dataset introduced above. The performance is evaluated using CalcED in both the strict mode and the relaxed mode. We also ran the original E2AP with a normal (Levenshtein) edit distance algorithm for comparison. As illustrated in Table II, E2AP, E2AP+ (strict), and E2AP+ (relaxed) all performed well with the test cases of the easy difficulty level, which is consistent with the findings from [24]. The algorithms also have the same accuracy in inferring the user activities for the easy level dataset because there is no ambiguity in it.

TABLE II: Evaluation results of E2AP and E2AP+ on the synth-new dataset.

Difficulty	Len	Accuracy		
		E2AP	E2AP+(strict)	E2AP+(relaxed)
Easy	387	0.9959	0.9959	0.9959
	1494	0.9959	0.9959	0.9959
	2959	0.9964	0.9964	0.9964
	10000	0.9959	0.9959	0.9959
Hard	387	0.6849	0.4310	0.9997
	1494	0.6848	0.4338	0.9994
	2959	0.6844	0.4339	0.9996
	10000	0.6857	0.4347	0.9997

As the difficulty level increases, the accuracies of E2AP+ in strict mode as well as the original E2AP drop significantly. For example, running the E2AP achieves the accuracy of 0.9959 on the dataset of easy level whose length is 10000, but only has an accuracy of 0.6857 on the dataset of hard difficulty with the same length. However, running E2AP+ in relaxed mode results in high accuracy for all difficulty levels and activity lengths.

This result does not mean using the relaxed mode of CalcED is always better, but it shows that, if our use case requires looking for all possible occurrences of a user activity, we can tolerate false positives in exchange for finding all potential occurrences of an activity. An example is the digital forensics in Section 4. In this situation, E2AP+ can provide substantially more information than E2AP.

C. Performance Evaluation of FindAct and WMatch

We implemented the FindAct algorithm and the WMatch algorithm and evaluated their performance on the synthetic dataset. We focused on Activity 1 to Activity 6 in [22] and [24], which relate to home security and share some common device events in their patterns. When comparing the inferred user activity sequence from the algorithms with the ground truth activity sequence, we allowed up to δ seconds of difference in their timestamps to account for asynchronous clocks, network latency, and variations in user behaviors.

For WMatch, we tested the algorithm using two sets of θ values. Here, because each kind of activity has different duration, we assign different θ values to each kind of activity. The θ value for activity A^i is determined by a quantile function $F_\Theta(\theta) := \Pr(\Theta_i \leq \theta_i) = p$. Here Θ_i is the random variable described in Section 5-A and p is the percentile parameter. Intuitively, a higher p value yields higher θ_i and a lower p value yields lower θ_i.

Table III illustrates the evaluation results of WMatch, FindAct, and IoTMosaic ($k \leq 5$) from [22] on the synthetic dataset with δ set to 1 second and 3 seconds. Numerical results are sums over a total of 100 test cases. The Correct column lists the number of occurrences of Activities 1-6 in the ground truth sequences that are correctly inferred by the respective algorithm, the Missed column lists the occurrences that the algorithm failed to infer, and the Recall column lists the ratio of correctly inferred activity occurrences over all activity occurrences in the ground truth. Note that for IoTMosaic both δ and p are not applicable, as its matching algorithm is different from WMatch and FindAct.

We can note two distinct trends in the results. First, the algorithms perform better in easy dataset compared to hard dataset. This conforms to our initial expectation as more ambiguity in the data adds to the challenges we discussed in Section 3. Second, it is apparent that a higher δ value significantly improves the recall of the WMatch and the FindAct algorithms. It should be noted that the outputs of the algorithms are not changed by different δ values. Instead, we change the "strictness" that determines whether a proposed match is accepted as a correct answer. However, we have to carefully decide on an appropriate δ because as δ gets larger, the results lose precision. For example, if we let $\delta = 60$ seconds, we allow matches up to 1 minute away from the ground truth to be counted as correct. Even if the recall may turn out to be very high for some cases, the results are too imprecise to be useful. For our digital forensics scenario, we consider $\delta = 3$ seconds to be satisfactory.

For WMatch, we note that it performs better at the higher p value, which is expected. As indicated above, higher p values yield higher θ_i values, which allows more matches to be found. The performance of WMatch is also consistent across both difficulties, with hard cases causing little degradation in the recall. It is important to understand that the higher the value of θ_i is, the more likely that WMatch finds a invalid match for activity i. Similar to δ, p needs to be set up appropriately to the situation in order to achieve a good balance between recall and accuracy. At $\delta = 3$ and $p = 0.9999$, we find that WMatch outperforms FindAct and IoTMosaic in both easy and hard datasets. At $\delta = 3$ and $p = 0.8$, we find that WMatch outperforms FindAct and IoTMosaic in hard test cases, but it performs slightly worse than the two algorithm in easy test cases.

Thus, we can conclude that all three algorithms can perform well on dataset with no ambiguity. For dataset with a lot of patterns sharing the same event sequence such as hard dataset, WMatch can still perform well under well-configured p and δ values.

6. Related Works

With heterogeneous IoT devices now deployed in smart homes [15], researchers from different communities have been working on analyzing the home network traffic [1, 7, 8, 11,

TABLE III: Evaluation results of WMatch with $p = 0.8$ and $p = 0.9999$, FindAct, and IoTMosaic ($k \leq 5$) on the synth-new dataset (sums over 100 cases).

Algorithm	Difficulty	Len	$\delta = 1$ Correct	$\delta = 1$ Missed	$\delta = 1$ Recall	$\delta = 3$ Correct	$\delta = 3$ Missed	$\delta = 3$ Recall
WMatch $p = 0.8$	Easy	387	1864	6864	0.2136	7522	1206	0.8618
		1494	7175	27039	0.2097	29535	4679	0.8632
		2959	14361	54042	0.2099	59017	9386	0.8628
		10000	47957	182468	0.2081	199081	31344	0.8640
	Hard	387	1788	6940	0.2049	7356	1372	0.8428
		1494	6973	27241	0.2038	28967	5247	0.8466
		2959	13815	54588	0.2020	57639	10764	0.8426
		10000	46921	183504	0.2036	194377	36048	0.8436
WMatch $p = 0.9999$	Easy	387	2266	6462	0.2596	8727	1	0.9999
		1494	8589	25625	0.2510	34210	4	0.9999
		2959	17300	51103	0.2529	68395	8	0.9999
		10000	57582	172843	0.2499	230397	28	0.9999
	Hard	387	2215	6513	0.2538	8728	0	1.0000
		1494	8497	25717	0.2483	34207	7	0.9998
		2959	17016	51387	0.2488	68395	8	0.9999
		10000	57713	172712	0.2505	230395	30	0.9999
FindAct	Easy	387	2041	6526	0.2382	7842	725	0.9154
		1494	7700	25811	0.2298	30690	2821	0.9158
		2959	15581	51454	0.2324	61450	5585	0.9167
		10000	51817	173895	0.2296	206745	18969	0.9160
	Hard	387	1185	7286	0.1399	4908	3563	0.5794
		1494	4506	28639	0.1359	18935	14210	0.5713
		2959	9169	57191	0.1382	38255	28105	0.5765
		10000	30698	192699	0.1374	127926	95471	0.5726
IoTMosaic $k \leq 5$	Easy	387	8675	53	0.9939	(δ does not apply to IoTMosaic)		
		1494	33966	248	0.9928			
		2959	67860	543	0.9921			
		10000	228764	1661	0.9928			
	Hard	387	3667	5061	0.4201			
		1494	14778	19436	0.4319			
		2959	29418	38985	0.4301			
		10000	99269	131156	0.4308			

12, 15, 17, 19–23, 25, 26], detecting design flaws in smart home IoT ecosystem [2, 9, 10, 13, 16, 18, 27], and security monitoring [3–6, 14].

The pervasive network traffic generated by a variety of smart home IoT devices has enabled deep understanding and multi-dimensional profiling of IoT devices' traffic patterns. A framework is proposed by [20] to collect all incoming, outgoing, and internal network traffic of IoT devices using programmable home routers. The traffic is further characterized and mined for extracting spatial, temporal, entropy, and cloud service patterns of IoT devices in edge networks. The authors of [15] built up smart home testbeds in laboratory environments to highlight the potential sensitive information exposures via analyzing the destinations of the network packets sent from the IoT devices.

The multidimensional traffic patterns extracted from the smart home network traffic have enabled many innovative applications such as device classification and anomaly detection. A recent study [25] notices that unencrypted information in the headers of the broadcasting and multicasting DHCP, SSDP, and mDNS messages sent out by IoT devices can be used to uniquely identify IoT device models. IoT SENTINEL [12], on the other hand, proposes a device type identification framework by building up fingerprints of each device type.

Recent studies on packet-level network traffic analysis have shown that it is possible to extract the concrete IoT device event logs. HoMonit [26] focuses on the Samsung's SmartThings platform to study encrypted wireless network traffic at the data link layer of IoT devices and analyze the mobile apps to infer the device events. Pingpong [19] identifies the *request* and *reply* communication patterns of IoT devices' network traffic, which helps to build up accurate signatures of IoT device events. Then state transition machines are applied by [19] for detecting the device event signatures in the collected network traffic trace. IoTAthena [21] further pinpoints the important inter-packet time interval feature in the device event signatures and designs a time-sensitive signature matching algorithm for efficient device event extraction.

With the availability of accurate device event logs from the smart home network traffic as well as the knowledge of device deployment information, some research efforts have been devoted to addressing the user activity inference problem. For example, Peek-a-Boo [1] first extracts the IoT device model and device state information from the sniffed wireless traffic, and then trains a Hidden Markov Model (HMM) with these features for inferring simple user activities involving a limited number of IoT devices. IoTMosaic [22] generates signatures for diverse user activities that are crucial to home safety, which consist of sequences of IoT device events. IoTMosaic [22] then designs efficient approximate matching algorithms to address the missing or out-of-order device events problems during the user activity inference. A recent work [24] formally defines the Events to Activities (E2A) and Events to

Activity Patterns (E2AP) problems and proposes a two-phase scheme where Phase 1 applies the AKMatch algorithm to compute a small number of representative matches of user activity patterns which facilitate faster processing without losing critical information. An unsupervised machine learning algorithm is designed and implemented by [24] in Phase 2 to match a compatible set of user activity patterns with optimized total weights in a finite number of iterations which is described in detail in Section 2.

Our work focuses on the user activity inference problem by identifying the challenges and limitations of the existing solutions. In particular, we point out that both [22] and [24] have poor performances when dealing with the ambiguities in user activity patterns and do not consider the important activity duration information. Subsequently, we highlight the opportunities in addressing these challenges and propose an extended framework with consideration of potential user activity pattern ambiguities and duration limitations and apply it to a digital forensics application.

7. Conclusions and Future Works

In this paper, we first pinpoint the challenges in inferring user activities from IoT device events faced by existing solutions and then analyze the root causes of their poor performances when applied to certain types of user activity patterns and device event sequences. We then explore the opportunities for extracting useful information even with the existence of ambiguities in user activity patterns. We propose an extension based on the state-of-the-art algorithm and then apply it to a digital forensics application. Our Future work will be centered on evaluating our solution with datasets generated in different smart home environments as well as exploring more approaches to tackle the challenges in user activity inference.

References

[1] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-Boo: I see your smart home activities, even encrypted!" in *Proc. of ACM WiSec*, 2020.

[2] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security evaluation of home-based IoT deployments," in *Proc. of IEEE S&P*, 2019.

[3] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *Proc. of ACM CCS*, 2019.

[4] W. Ding and H. Hu, "On the safety of IoT device physical interaction control," in *Proc. of ACM CCS*, 2018.

[5] W. Ding, H. Hu, and L. Cheng, "IOTSAFE: Enforcing safety and security policy with real IoT physical interaction discovery," in *Proc. of NDSS*, 2021.

[6] C. Fu, Q. Zeng, and X. Du, "HAWatcher: Semantics-aware anomaly detection for appified smart homes," in *Proc. of USENIX Security*, 2021.

[7] S. Hui, H. Wang, D. Xu, J. Wu, Y. Li, and D. Jin, "Distinguishing between smartphones and IoT devices via network traffic," *IEEE Internet of Things Journal*, vol. 9, pp. 1182–1196, 2022.

[8] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "IoT devices fingerprinting using deep learning," in *Proc. of IEEE MILCOM*, 2018.

[9] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds," in *Proc. of IEEE S&P*, 2020.

[10] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of IoT devices on home networks," in *Proc. of USENIX Security*, 2019.

[11] X. Ma, J. Qu, J. Li, J. C. Lui, Z. Li, and X. Guan, "Pinpointing hidden IoT devices via spatial-temporal traffic fingerprinting," in *Proc. of IEEE INFOCOM*, 2020.

[12] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated device-type identification for security enforcement in IoT," in *Proc. of IEEE ICDCS*, 2017.

[13] P. Morgner, S. Mattejat, Z. Benenson, C. Müller, and F. Armknecht, "Insecure to the touch: Attacking ZigBee 3.0 via touchlink commissioning," in *Proc. of ACM WiSec*, 2017.

[14] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "DÏoT: A federated self-learning anomaly detection system for IoT," in *Proc. of IEEE ICDCS*, 2019.

[15] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. of ACM IMC*, 2019.

[16] E. Ronen, C. O'Flynn, A. Shamir, and A.-O. Weingarten, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Proc. of IEEE S&P*, 2017.

[17] 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, pp. 1745–1759, 2018.

[18] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smart-phones attacking smart-homes," in *Proc. of ACM WiSec*, 2016.

[19] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "PingPong: Packet-level signatures for smart home device events," in *Proc. of NDSS*, 2019.

[20] Y. Wan, K. Xu, F. Wang, and G. Xue, "Characterizing and mining traffic patterns of IoT devices in edge networks," *IEEE Transactions on Network Science and Engineering*, vol. 8, pp. 89–101, 2021.

[21] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTAthena: Unveiling IoT device activities from network traffic," *IEEE Transactions on Wireless Communications*, vol. 21, pp. 651–664, 2022.

[22] Y. Wan, K. Xu, F. Wang, and G. Xue, "IoTMosaic: Inferring user activities from IoT network traffic in smart homes," in *Proc. of IEEE INFOCOM*, 2022.

[23] A. Wang, A. Mohaisen, and S. Chen, "XLF: A cross-layer framework to secure the Internet of Things (IoT)," in *Proc. of IEEE ICDCS*, 2019.

[24] G. Xue, Y. Wan, X. Lin, K. Xu, and F. Wang, "An effective machine learning based algorithm for inferring user activities from IoT device events," *IEEE JSAC Series on Machine Learning for Communications and Network*, 2022, accepted.

[25] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and IoT devices from (public) WiFi," in *Proc. of USENIX Security*, 2020.

[26] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit: Monitoring smart home apps from encrypted traffic," in *Proc. of ACM CCS*, 2018.

[27] W. Zhou, Y. Jia, Y. Yao, L. Zhu, L. Guan, Y. Mao, P. Liu, and Y. Zhang, "Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms," in *Proc. of USENIX Security*, 2019.