GraphTrack: An unsupervised graph-based cross-device tracking framework

Tianchen Zhou
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd
Part of the Computer Sciences Commons

Recommended Citation
Zhou, Tianchen, "GraphTrack: An unsupervised graph-based cross-device tracking framework" (2018). Graduate Theses and Dissertations. 17377.
https://lib.dr.iastate.edu/etd/17377

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
GraphTrack: An unsupervised graph-based cross-device tracking framework

by

Tianchen Zhou

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Neil Zhenqiang Gong, Co-major Professor
Jin Tian, Co-major Professor
Wensheng Zhang

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2019

Copyright © Tianchen Zhou, 2019. All rights reserved.
DEDICATION

To my parents.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2. RELATED WORK</td>
<td>6</td>
</tr>
<tr>
<td>CHAPTER 3. PROBLEM DEFINITION</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Mobile-Desktop Tracking</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Design Goals</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER 4. GRAPHTRACK</td>
<td>12</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>12</td>
</tr>
<tr>
<td>4.2 Unsupervised Mobile-Desktop Tracking Framework</td>
<td>12</td>
</tr>
<tr>
<td>4.2.1 Modeling IPs, Domains, and Devices via Graphs</td>
<td>13</td>
</tr>
<tr>
<td>4.2.2 Modeling Device Similarity using Random Walk with Restart on Graphs</td>
<td>14</td>
</tr>
<tr>
<td>4.2.3 A Running Example</td>
<td>17</td>
</tr>
<tr>
<td>4.2.4 GraphTrack using IPs or Domains Alone</td>
<td>19</td>
</tr>
<tr>
<td>4.2.5 Combining IPs and Domains</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Incorporating Manual Labels</td>
<td>21</td>
</tr>
<tr>
<td>4.4 Computational Complexity</td>
<td>22</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 5.1 Results of BAT-Raw, the implemented BAT-SU, and GraphTrack-OR-SU on the same training set and testing set as in Zimmeck et al. (2017). BAT-Raw has 37 TPs, 5 FPs, 0 TN, and 2 FNs; BAT-SU has 36 TPs, 6 FPs, 0 TN, and 2 FNs; GraphTrack-OR-SU has 39 TPs, 2 FPs, 0 TN, and 3 FNs. 35
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>(a) An example IP-Device graph. (b) An example Domain-Device graph.</td>
<td>13</td>
</tr>
<tr>
<td>4.2</td>
<td>An example Domain-Device graph to illustrate how adapted RWwR can capture latent correlations, while the state-of-the-art method cannot.</td>
<td>17</td>
</tr>
<tr>
<td>5.1</td>
<td>Performance of GraphTrack-Domain with and without the top-50 domains.</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparing mobile-desktop tracking methods using IPs or domains alone.</td>
<td>28</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparing different ways to combine IPs and domains.</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>Impact of the fraction of unmatched mobile devices on the performance of GraphTrack-OR.</td>
<td>30</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparing GraphTrack-OR with BAT.</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>Impact of error rates in single-device tracking on Accuracy of cross-device tracking.</td>
<td>32</td>
</tr>
<tr>
<td>5.7</td>
<td>Alternatives of converting BAT-SU to be unsupervised. (a) Accuracy vs. threshold. (b) Gaps between top two similarity scores for 20 sampled mobile devices.</td>
<td>33</td>
</tr>
<tr>
<td>5.8</td>
<td>Comparing GraphTrack with the state-of-the-art method, where (a) labeled device pairs are randomly sampled and (b) labeled device pairs are obtained via cross-device IDs on a single domain.</td>
<td>36</td>
</tr>
<tr>
<td>5.9</td>
<td>(a) Running time of unsupervise methods. (b) Running time of supervised methods.</td>
<td>37</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Neil zhengqiang Gong for his guidance, patience and support throughout this research and the writing of this thesis. I would also like to thank my co-worker Binghui without whom I cannot finish this work by my own. Last but not least I would like to thank my parents and my boyfriend for their loving guidance and supports.
ABSTRACT

Cross-device tracking has drawn growing attention from both commercial companies and the general public because of its privacy implications as well as applications for user profiling, personalized services, and user authentication. One particular, widely-used type of cross-device tracking is to leverage browsing histories of user devices, e.g., characterized by a list of IP addresses used by the devices and domains visited by the devices. State-of-the-art browsing history based methods compute a similarity score for a device pair using only the common IPs used by both devices and domains visited by both devices, and leverage supervised machine learning. These methods cannot capture latent correlations among IPs/domains and require a large amount of labeled device pairs, which is time-consuming and costly to obtain.

In this work, GraphTrack, an unsupervised graph-based cross-device tracking framework, to track users across different devices by correlating browsing histories on these devices is proposed. Specifically, the complex interplays among IPs, domains, and devices are modeled as graphs to capture the latent correlations between IPs/domains. Moreover, random walk with restart is adapted to compute similarity scores between devices based on the graphs. GraphTrack leverages the similarity scores to perform unsupervised cross-device tracking and can be extended to incorporate manual labels if available. GraphTrack is evaluated on a real-world dataset. The results show that GraphTrack substantially outperforms the state-of-the-art method, e.g., by 13% in Accuracy.
CHAPTER 1. INTRODUCTION

Cross-device tracking—a technique used to identify owners of various devices, such as mobile phones and desktops—has drawn much attention of both commercial companies and the general public. For example, Drawbridge dra (2017), an advertising company, goes beyond traditional device tracking to identify users behind different devices. Due to the increasing demands and corresponding privacy concerns, the U.S. Federal Trade Commission hosted a workshop Commission (2015) in 2015 and released a staff report Commission (2017) about cross-device tracking and industry regulations in early 2017. The growing interest in cross-device tracking, or device tracking in general, is highlighted by the privacy implications associated with tracking as well as the applications of tracking for user profiling, personalized services, and user authentication. For example, a bank application can adopt cross-device tracking as a part of multi-factor authentication to increase account security.

In the past, cross-device tracking mainly leverages cross-device IDs, background environment, or browsing history of the devices. For instance, cross-device IDs may include a user’s email address or username, which are not applicable for third-party trackers and when users do not register accounts or do not login. Background environment (e.g., ultrasound Mavroudis et al. (2017)) also cannot be applied when devices are used in different environments such as home and workplace. Therefore, we focus on browsing history based cross-device tracking Zimmeck et al. (2017); Malloy et al. (2017).

Specifically, a browsing history based tracking utilizes source and destination pairs—e.g. the client IP address and the destination website’s domain name—of users’ browsing records to correlate different devices of the same user. Several browsing history based cross-device tracking methods Zimmeck et al. (2017); Walthers (2015); Selsaas et al. (2015); Anand and Renov (2015); Díaz-Morales (2015); Landry et al. (2015); Kim et al. (2015); Kejela and Rong (2015); Cao et al. (2015); Malloy et al. (2017); dra (2017); cri (2017) have been proposed recently. For instance, Zim-
meck et al. Zimmeck et al. (2017) proposed a supervised cross-device tracking method (called BAT for simplicity) that achieves state-of-the-art performance. BAT computes a similarity score via Bhattacharyya coefficient Wang and Pu (2013) for a pair of devices based on the common IPs used by both devices and domains visited by both devices. Given a set of training device pairs, BAT determines a threshold of similarity score. Then, given a device $D$, BAT computes the similarity score between $D$ and every other device; BAT predicts $D$ to match the device that has the largest similarity score with $D$, if the similarity score is larger than the threshold.

The major limitation of existing browsing history based methods is that they cannot capture the latent correlations between domains/IPs in the browsing history. Let us first look at the domain, i.e., the destination in the browsing history. Suppose both Facebook and Twitter are frequently visited by a large amount of devices, and thus there exists a certain latent correlation between them (in this example, both of them are social media sites). Assume a user visits Facebook frequently on one device but visits Twitter frequently on another device. The two devices could have a large similarity score because they both frequently visit sites that have latent correlations. However, state-of-the-art methods like BAT-SU would compute a very low similarity score for the two devices because BAT-SU only leverages the common domains visited by both devices to compute similarity scores. Likewise, IP addresses of devices, i.e., the sources in the browsing history, could have latent correlations. For example, say many users travel between a dormitory and a campus in a daily basis. Thus, a dormitory IP on one device and a campus IP on another should produce a large similarity score between these two devices, but BAT-SU fails to do so.

**Our work** In this thesis, *GraphTrack*, a graph-based framework, is proposed to perform cross-device tracking that captures latent correlations among IPs/domains. For instance, in the above example of a user visiting Facebook on one device and Twitter on another device; once a large number of other users visit both Facebook and Twitter on a same device, GraphTrack can leverage such crowd data to discover the latent relationships between Facebook and Twitter, and assign a high similarity score for the two devices.
Specifically, the complex interplays are leveraged between devices and browsing histories of many users to capture the latent correlations. The intuition is that if many devices visit two domains (or use two IPs), then there could be a latent relationship between them. The complex interactions between IPs, domains, and devices are proposed to be modeled as graphs to capture such intuition; and random walk with restart (RWwR) is adapted on the graphs to compute similarity scores between devices, which captures the latent correlations between domains/IPS. Specifically, an IP-Device graph is proposed to model the interactions between IPs and devices, and a Domain-Device graph is proposed to model interactions between domains and devices. IPs and domains are considered separately because they are different data types. In the IP-Device (or Domain-Device) graph, a node means an IP (or domain) and an edge means the corresponding device used the corresponding IP (or visited the corresponding domain). BAT is noted essentially uses the number of common neighbors (with certain normalization) of two devices in proposed graphs to compute the similarity score between the two devices. GraphTrack leverages RWwR, which can better capture the graph structure, to compute similarity scores.

The proposed GraphTrack has the following properties:

- Adaptiveness to small amount or even no training data: Training samples are scarce for the tracking purpose especially for third-party trackers who do not have access to any ground-truth data. In the absence of training data, GraphTrack leverages the similarity scores computed based on the graphs to perform unsupervised cross-device tracking. Intuitively, if two devices A and B are used by the same user, then A is very likely to have the largest similarity score with B among all the devices and vice versa. GraphTrack leverages such symmetry to perform unsupervised tracking. Specifically, GraphTrack predicts that two devices are matched only if one device is the most similar to the other device among all devices and vice versa. Moreover, when manually labeled device pairs are available, we can extend GraphTrack to incorporate them. Specifically, the labeled device pairs are used to learn a similarity threshold, and two devices are predicted matched if their similarity score is larger than the threshold.
Robustness to uncertainties in single-device tracking: Cross-device tracking relies on single-device tracking, which links browsing history to a device. In some scenarios, e.g., tracking using browser fingerprinting Yen et al. (2012); Nikiforakis et al. (2013); Acar et al. (2014, 2013); Laperdrix et al. (2016); Boda et al. (2012); Englehardt and Narayanan (2016), single-device tracking could have uncertainty/errors at linking browsing history to devices. GraphTrack models the weight of an edge as the number of times that the device used the IP (or visited the domain). For instance, suppose a device visited a domain multiple times; once a single-device tracker links a majority of them to the device, the associated edge still has a large weight. However, if a device did not visit a certain domain, but the single-device tracker occasionally links the domain with the device, then the weight of the corresponding edge is small. Thus, the incorrectly linked domain has a small impact on the overall structure of the Domain-Device graph. The evaluation shows that GraphTrack’s Accuracy is still 72% when the single-device tracker incorrectly links 10% of each device’s browsing history to random devices. However, with the same uncertainty, BAT only achieves an Accuracy of 48%.

High correlation accuracy: GraphTrack is evaluated against BAT under the same setting and GraphTrack is shown constantly outperforms BAT—for example, GraphTrack outperforms BAT by 28% in terms of accuracy when 30% of data are labelled. Because BAT cannot track users when no training data is available, BAT is also adapted to be an unsupervised method, called BAT-uSU using GraphTrack framework. The evaluation shows that GraphTrack outperforms the unsupervised version by 13% in Accuracy.

Key contributions are summarized as follows:

- GraphTrack, a graph-based framework, is proposed to perform cross-device tracking. GraphTrack is unsupervised but can incorporate manually labeled device pairs if available; GraphTrack can capture the latent correlations between domains/IPs; and GraphTrack is robust to uncertainty in single-device tracking.
- Complex interplays between IPs, domains, and devices are modeled as graphs. Moreover, RWwR is adapted to analyze the structure of the graphs and capture latent correlations among IPs and domains.

- GraphTrack is evaluated and compared with state-of-the-art method using a publicly available real-world dataset. The results demonstrate that GraphTrack significantly outperforms the state-of-the-art method.
CHAPTER 2. RELATED WORK

Single-device tracking It refers to techniques that are used to identify a single device, such as a desktop, a mobile phone, and a tablet. Prior work on single-device tracking can be roughly classified into two categories: cookie or super-cookie based and browser fingerprinting. First, Roesner et al. Roesner et al. (2012) surveyed and measured top Alexa websites and identified a significant number of trackers in the wild. Lerner et al. Lerner et al. (2016) conducted an archaeological study of web tracking using Internet time machine to understand the evolution of tracking from 1996 to 2016. Metwalley et al. Metwalley and Traverso (2015) measured web tracking using an unsupervised method. Other than measuring the significance of tracking, some research work Krishnamurthy and Wills (2006, 2009); Krishnamurthy et al. (2011); Krishnamurthy and Wills (2008); Mayer and Mitchell (2012); Sánchez-Rola et al. (2015) studied the privacy implication of web tracking, such as business model and the leak of email addresses and user names.

Second, several studies Yen et al. (2012); Nikiforakis et al. (2013); Acar et al. (2014, 2013); Laperdrix et al. (2016); Boda et al. (2012); Englehardt and Narayanan (2016) performed measurement studies on browser fingerprinting, a second-generation web tracking technique that utilizes browser features such as number of plugins, fonts, and user agents. Fifield et al. Fifield and Egelman (2015) focused on a specific feature of browser, i.e., fonts, and proposed to use a subset of fonts for browser fingerprinting. Mowery et al. Mowery and Shacham (2012) proposed to use canvas, a HTML5 feature, for fingerprinting. Mulazzani et al. Mulazzani et al. (2013) and Mowery et al. Mowery et al. (2011) fingerprinted browsers using features of JavaScript engine. Nakibly et al. Nakibly et al. (2015) proposed several tracking techniques using features from hardware including microphone, motion sensor, and GPU. Cao et al. Cao et al. (2017b) extended existing browser fingerprinting techniques to be cross-browser, e.g., among IE, Firefox, and Chrome.
Single-device tracking is the basis of cross-device tracking, i.e., cross-device tracking needs to identify single device first and then link devices together. Cross-device tracking also goes beyond single-device tracking, because cross-device tracking can identify the user behind the devices.

**Cross-device tracking** It is a relatively new research area and refers to techniques that are used to identify users of different devices. In 2015, Drawbridge released a challenge dra (2015) to the research community about browsing history based cross-device tracking; and then multiple research papers were published on the topic of cross-device tracking Walthers (2015); Selsaas et al. (2015); Anand and Renov (2015); Díaz-Morales (2015); Landry et al. (2015); Kim et al. (2015); Kejela and Rong (2015); Cao et al. (2015). Zimmeck et al. Zimmeck et al. (2017) recently proposed another browsing history based supervised cross-device tracking method, which achieved state-of-the-art performance. Malloy et al. Malloy et al. (2017) proposed a device graph based on IP colocations. Specifically, in their device graph, a node is a device and an edge is created between two devices if they used the same IP address. However, such device graph does not distinguish the IPs that are commonly used by devices, but different IPs have different impact on the similarity between devices. IP-Device graph distinguishes between different IPs by adding each IP address as a node in the graph. Moreover, devices in a connected component or community of a device graph are predicted to belong to the same user or household dra (2017); cri (2017); Malloy et al. (2017). However, such graph analysis techniques are not applicable to IP-Device graph or Domain-Device graph. Apart from browsing history based cross-device tracking, ultrasound was also proposed to link different devices Mavroudis et al. (2016, 2017) when they are positioned in the same environment. By contrast, browsing history based cross-device tracking can link two devices even when they are in different environments, such as at home and in the workplace. Brookman et al. Brookman et al. (2017) conducted a measurement study about cross-device tracking in the wild.

**Defense against single-device tracking** Because tracking violates user privacy, many methods have been proposed to defend against device tracking. Existing defenses focus on single-device tracking and can be classified into two categories: defense against cookie and super-cookie based tracking and defense against browser fingerprinting. First, ShareMeNot Roesner et al. (2012);
vate browsing mode Wikipedia (2017); Xu et al. (2015), TrackingFree Pan et al. (2015), and Meng et al. Meng et al. (2016) are examples of defending against cookie and super-cookie based tracking. The key idea of such approaches is to isolate tracking entities into different units and prevent tracking. Second, FuzzyFox Kohlbrenner and Shacham (2016) and PriVaricator Nikiforakis et al. (2015) proposed adding noise to browsers so that the fingerprint will change all the time. On the contrary, DeterFox Cao et al. (2017a) and Tor Browser Perry et al. (2015) normalize the outputs so that the fingerprinting results remain the same across different devices. Although many defenses have been proposed, device and cross-device tracking is still popular in the current Internet landscape.
CHAPTER 3. PROBLEM DEFINITION

3.1 Mobile-Desktop Tracking

Cross-device tracking can be mobile-desktop tracking, desktop-mobile tracking, desktop-desktop tracking, and mobile-mobile tracking. For simplicity, we focus on mobile-desktop tracking, since it was empirically observed in real-world Zimmeck et al. (2017). However, GraphTrack can also be applied to other types of cross-device tracking. More details are discussed in Section 6.

Threat model Both first-party tracker and third-party tracker are considered. A tracker is a first-party tracker if the tracker tracks the users who visit its web services. For instance, Facebook is a first-party tracker when tracking users who use Facebook. A tracker is a third-party tracker if the tracker tracks users who visit other parties’ web services. Example third-party trackers could be an ads domain like doubleclick, a social network add-on like Facebook like button, and a third-party JavaScript library like Google Analytics. Such third-party trackers can collect users’ browsing history via, for example, the referer header upon the websites that embed such trackers. According to existing surveys Roesner et al. (2012), more than 90% of Alexa Top websites include at least one third-party tracker, and some third-party trackers even collaborate with each other.

Suppose a tracker has collected a large amount of pairs \((IP, Domain)\) from many devices at different time, where a pair \((IP, Domain)\) from a device means that the device once visited \(Domain\) using the \(IP\) address. The tracker first links the pairs \((IP, Domain)\) to devices, which is known as single-device tracking. Various techniques—such as cookies or super-cookies Roesner et al. (2012), browser fingerprinting Yen et al. (2012); Nikiforakis et al. (2013); Acar et al. (2014, 2013); Laperdrix et al. (2016); Boda et al. (2012); Englehardt and Narayanan (2016), and cross-browser fingerprinting Boda et al. (2012); Cao et al. (2017b)—can be used to perform single-device tracking. In this thesis it is assumed that single-device tracking has been done by existing works, which have already distinguished mobile devices and desktop devices. Now, suppose we have \(m\)
mobile devices and \( n \) desktop devices. After single-device tracking, we have a browsing history \( \{(IP_1, Domain_1), \cdots, (IP_k, Domain_k)\} \) for each device. Note that the same pair \((IP, Domain)\) could appear multiple times in the browsing history of a device.

The tracker’s goal is to predict whether each mobile device and certain desktop device belong to the same user. If yes, the tracker can combine the IPs and domains used/visited by both devices to better profile the user, e.g., infer users’ demographics and interests with higher accuracies Zimmeck et al. (2017), which in turn helps deliver targeted advertisements to users. Formally, define the mobile-desktop tracking problem as follows:

**Definition 1 (Mobile-Desktop Tracking)** Suppose we are given \( m \) mobile devices \( \mathcal{M} = \{M_1, \cdots, M_m\} \), \( n \) desktop devices \( \mathcal{D} = \{D_1, \cdots, D_n\} \), and a browsing history of each device produced by single-device tracking. Mobile-desktop tracking is to design a mapping \( T : \mathcal{M} \rightarrow \mathcal{D} \cup \perp \). Specifically, \( T(M_i) = D_j \) means the tracker predicts the mobile device \( M_i \) and the desktop device \( D_j \) belong to the same user; and \( T(M_i) = \perp \) means the tracker predicts no desktop device in \( \mathcal{D} \) and the mobile device \( M_i \) belong to the same user.

When a mobile device and a desktop device belong to the same user, we call the mobile device matches the desktop device. When no desktop device and a mobile device belong to the same user, we call the mobile device has no match.

### 3.2 Design Goals

A mobile-desktop tracking method is designed to achieve the following goals.

1) **Leveraging both IPs and domains.** IPs and domains are complementary information sources. Specifically, IPs used by devices represent their geolocations, while visited domains could indicate interests of the users. Therefore, when both IPs used by devices and domains visited by devices are available, GraphTrack should leverage both of them to enhance tracking performance. However, GraphTrack should also be applicable when only IPs or domains are available, e.g., a first-party tracker may only be able to collect IPs used by devices.
2) **Capturing latent correlations among IPs/domains.** IPs used by devices (or domains visited by devices) could have latent correlations. GraphTrack should discover such latent correlations and leverage them to track devices.

3) **Without requiring labeled device pairs.** In some scenarios, the tracker may be able to obtain some labeled device pairs. For instance, a first-party tracker could use cross-device IDs to obtain labeled device pairs from the users who log in the tracker’s web services on multiple devices. However, in some other scenarios, it is challenging for the tracker to obtain labeled device pairs. For instance, a third-party tracker can hardly obtain labeled device pairs. Therefore, GraphTrack should be applicable with or without labeled device pairs.

4) **Robust to uncertainty in single-device tracking.** Mobile-desktop tracking relies on single-device tracking to reliably link the browsing histories to devices. However, single-device tracking (especially tracking using browser fingerprinting) often has uncertainty, e.g., an IP used by one device or a domain visited by one device is incorrectly linked to another device. For instance, if cookie or super-cookie based single-device tracking is adopted, a user may clear web cookies, or switch to private browsing mode or another browser, so that browsing histories of the same device may be linked to two different ones. Moreover, browser fingerprinting, a new-generation of single-device tracking widely adopted by top Alexa websites Englehardt and Narayanan (2016), is unreliable itself. According to the most recent large-scale study, i.e., amIUnique Laperdrix et al. (2016), the accuracy of browser fingerprinting is only 89.4%. That is, two devices may share the same fingerprinting and browsing histories of these two devices may be linked to one. Therefore, GraphTrack should be robust to uncertainty in single-device tracking.

State-of-the-art browsing history based methods (e.g., Zimmeck et al. (2017)) do not satisfy requirements 2), 3), and 4).
CHAPTER 4. GRAPHTRACK

4.1 Overview

The first challenge for mobile-desktop tracking is how to leverage heterogeneous data sources, i.e., IPs and domains, to identify the correlations between devices. The challenge is addressed by modeling the interplays between IPs, domains, and devices as graphs, and leveraging graph mining techniques (in particular random walk with restart on graphs) to capture the similarities between devices. The second challenge is how to match mobile devices to desktop devices without manual labels, i.e., in an unsupervised fashion. It is addressed by leveraging the symmetry between devices. Specifically, GraphTrack predicts that a mobile device matches a desktop device only if the desktop device is the most similar to the mobile device among all desktop devices and vice versa, where the similarity between devices is computed via analyzing the graph structure. The third challenge is how to be robust to uncertainty in single-device tracking that links browsing histories to devices. The challenge is addressed by leveraging the frequency that a device used a certain IP or visited a certain domain.

Next, GraphTrack is firstly discussed to perform unsupervised mobile-desktop tracking. Then, it is discussed how to incorporate manual labels into GraphTrack if they are available. Finally, the computational complexity of unsupervised and supervised GraphTrack methods is analyzed.

4.2 Unsupervised Mobile-Desktop Tracking Framework

First, the interplays between IPs (or domains) and devices are modeled as graphs. Second, random walk with restart, a popular graph mining technique is adapted to graphs to model similarities between devices. Third, mobile-desktop tracking is discussed using only IPs or domains based on the device similarities. Fourth, IPs and domains are combined for mobile-desktop tracking.
4.2.1 Modeling IPs, Domains, and Devices via Graphs

*IP-Device graph* and *Domain-Device graph* are proposed to model the interplays between IPs, domains, and devices. Then, random walks with restart are leveraged on the graphs to compute similarity scores between devices.

**Modeling interplays between IPs and devices as an IP-Device graph** Each unique IP address (or IP prefix), mobile device, and desktop device is represented as a node. An edge is created between an IP node and a device if the device used the IP at least once. Moreover, the edge weight of an edge between a device and an IP is modeled as the number of times that the device used the IP in its browsing history. This weighted graph is called *IP-Device graph*. Figure 4.1a shows an example IP-Device graph. Note that there are no edges between mobile devices and desktop devices. GraphTrack leverages the edge weights in the IP-Device graph to be more robust to uncertainty of single-device tracking at linking IPs to devices. For instance, suppose a device used an IP multiple times in its browsing history; once a single-device tracker links a majority of them to the device, the corresponding edge still has a large weight. However, suppose a device did not use a certain IP, but the single-device tracker occasionally links the IP with the device. Then, the weight of the corresponding edge is small, and thus the incorrectly linked IP has a small impact on the overall structure of the IP-Device graph.
Modeling interplays between domains and devices as a Domain-Device graph Each unique domain, mobile device, and desktop device is represented as a node; an edge between a domain and a device is created if the device visited the domain at least once; and similar to the IP-Device graph, the weight of an edge between a device and a domain is modeled as the number of times that the device visited the domain. This weighted graph is called Domain-Device graph. Figure 4.1b shows an example.

4.2.2 Modeling Device Similarity using Random Walk with Restart on Graphs

Random walk with restart (RWwR) Tong et al. (2006) (also known as Personalized PageRank Brin and Page (1998)) is leveraged to analyze the structure of the IP-Device graph and Domain-Device graph and model similarity between devices. A larger similarity score indicates a higher likelihood of match. Next, RWwR is firstly introduced on a general weighted graph. Then, RWwR is adapted to compute similarity scores between devices in IP-Device and Domain-Device graphs. Finally, an example is used to illustrate that adapted RWwR can capture latent correlations among domains, while state-of-the-art cannot.

Random walk with restart (RWwR) on graphs Suppose we have an undirected weighted graph $G = (V, E, W)$, where $V$, $E$, and $W$ are the set of nodes, edges, and edge weights, respectively. For instance, $(u, v)$ is an edge between nodes $u$ and $v$, while $w_{u,v}$ is the weight of the edge $(u, v)$. We perform an RWwR in the graph from a seed node $u$. Specifically, in an RWwR, we have a particle that can stay on nodes in the graph. Initially, the particle stays on node $u$. The particle iteratively moves along edges in the graph or jumps back to the initial node $u$. Suppose in the $t$th step, the particle stays on node $v$. In the $(t + 1)$th step, the particle jumps back to the initial node $u$ with a certain probability $\alpha$ (i.e., the random walk is restarted); and with the remaining probability $1 - \alpha$, the particle picks a neighbor $x$ of $v$ with a probability proportional to the edge weight $w_{x,v}$ and moves to the neighbor. $\alpha$ is called restart probability. Suppose $p_{u,v}$ is the frequency that the particle stays on node $v$. When the RWwR repeats for a large number of steps, $p_{u,v}$ becomes the probability that the particle will stay on node $v$ in each step. Conventionally, $p_u = [p_{u,1}, p_{u,2}, \cdots, p_{u,|V|}]$ is
called the *stationary distribution* of the RWwR. $p_{u,v}$ is a natural metric to measure similarity between nodes $u$ and $v$. A larger $p_{u,v}$ indicates $v$ is structurally closer to $u$ on the graph and thus $v$ is more similar to $u$. Such RWwR based similarity was applied to rank relevant webpages in search engines Brin and Page (1998), recommend user accounts they wish to follow in Twitter Gupta et al. (2013), detect spammers in social networks Yang et al. (2012), and infer user attributes in social networks Gong and Liu (2016).

Many techniques (e.g., Tong et al. (2006); Wang et al. (2017)) have been developed to compute the stationary distribution $p_u$ efficiently. For instance, $p_u$ can be iteratively computed as follows:

$$p_u^{(t+1)} = (1 - \alpha)Ap_u^{(t)} + \alpha 1_u,$$

(4.1)

where $p_u^{(t)} = [p_u^{(t)}(1), p_u^{(t)}(2), \cdots, p_u^{(t)}(|V|)]$ is the probability distribution of the RWwR in the $t$th iteration, $A$ is the transition matrix of the graph, and $1_u$ is an unit vector whose $u$th entry is 1 and other entries are 0. The transition matrix has $|V|$ rows and $|V|$ columns. Denote $\Gamma_u$ as the set of neighbors of $u$ and $d_v$ is the weighted degree of $v$, i.e., $d_v = \sum_{s \in \Gamma_u} w_{s,v}$. The $(u,v)$th entry of the transition matrix is formally defined as:

$$A_{u,v} = \begin{cases} \frac{w_{u,v}}{d_v} & \text{if } v \in \Gamma_u; \\ 0 & \text{otherwise}, \end{cases}$$

(4.2)

To compute $p_u$, initialize a random vector $p_u^{(0)}$ and then iteratively apply Equation 4.1 until the difference in two consecutive iterations is smaller than a certain threshold, e.g., $|p_u^{(t+1)} - p_u^{(t)}|_1 < 10^{-3}$.

**Adapting RWwR to IP-Device and Domain-Device graphs to compute similarity scores between devices** State-of-the-art method Zimmeck et al. (2017) computes the similarity scores between devices via Bhattacharyya coefficient, which is a *simple* weighted common neighbor metric Murata and Moriyasu (2007) and weights are the normalized frequencies of IPs or domains. As a result, Bhattacharyya coefficient is unable to capture latent correlations among IPs and domains. In contrast, RWwR on graphs is adapted to model similarity between mobile and desktop devices, which can capture latent correlations among IPs and domains. Specifically, an *IP-based similarity score* (denoted as $s_{IP}(M_i, D_j)$) between a mobile device $M_i$ and a desktop device $D_j$ is computed using an RWwR on the IP-Device graph; and a *domain-based similarity score* (denoted
as $s_{DO}(M_i, D_j)$ between a mobile device $M_i$ and a desktop device $D_j$ is computed using an RWwR on the Domain-Device graph.

Take computing IP-based similarity scores as an example to illustrate more details. Suppose we are given an IP-Device graph and a mobile device $M_i$, we aim to compute similarity scores between $M_i$ and each desktop device. One way is to simply apply an RWwR from $M_i$ in the weighted IP-Device graph, compute the stationary distribution $\mathbf{p}_{M_i}$ of the RWwR, and define the IP-based similarity score $s_{IP}(M_i, D_j)$ between $M_i$ and a desktop device $D_j$ as $s_{IP}(M_i, D_j) = p_{M_i, D_j}$, for any $D_j \in \mathcal{D}$. However, as demonstrated in experiments, such RWwR based similarity scores achieve suboptimal performance. This is because such similarity scores are heavily influenced by the weighted degree of a device, i.e., the total number of IP visits of a device. For instance, suppose a desktop device belongs to a heavy Internet user and has a dense browsing history. Thus, in the IP-Device graph, many edges of the desktop device have large weights. As a result, when the particle in the RWwR stays on an IP node, the particle is more likely to move from the IP node to the desktop device compared to other desktop devices, which means the desktop device will have a larger stationary probability and thus a larger similarity score with $M_i$. However, such similarity score is heavily biased by the weighted degrees of devices.

Normalizing the weight of an edge by the weighted degree of the corresponding device can address the issue. Specifically, each edge in the IP-Device graph connects an IP and a device. Suppose an edge $(x, y)$ connects an IP $x$ and a device $y$. Define a normalized edge weight $w'_{x,y}$ as

$$w'_{x,y} = \frac{w_{x,y}}{\sum_{z \in \Gamma_y} w_{z,y}}. \quad (4.3)$$

Then, define the transition matrix $\mathbf{A}$ in an RWwR using the normalized edge weights, i.e., replacing $w_{u,v}$ as $w'_{u,v}$ and $d_v$ as the sum of the normalized weights of edges of $v$ in Equation 4.2. In order to compute IP-based similarity scores between a mobile device $M_i$ and each desktop device, perform an RWwR from $M_i$ in the IP-Device graph using the adapted transition matrix. Define the IP-based similarity score $s_{IP}(M_i, D_j)$ between $M_i$ and a desktop device $D_j$ as $s_{IP}(M_i, D_j) = p_{M_i, D_j}$, where $\mathbf{p}_{M_i}$ is the stationary distribution of the RWwR. A larger similarity score $s_{IP}(M_i, D_j)$ indicates
that the desktop device $D_j$ is structurally closer to the $M_i$ in the IP-Device graph and thus $D_j$ is more likely to match $M_i$.

Moreover, RWwR is further leveraged to compute similarity scores between a desktop device and each mobile device. Specifically, perform an RWwR from $D_j$ in the IP-Device graph with the adapted transition matrix and compute the stationary distribution $p_{D_j}$ of the RWwR. Then, define the IP-based similarity score $s_{IP}(D_j, M_i)$ between a desktop device $D_j$ and a mobile device $M_i$ as $s_{IP}(D_j, M_i) = p_{D_j, M_i}$, for any $M_i \in M$. Note that $s_{IP}(M_i, D_j)$ and $s_{IP}(D_j, M_i)$ are very likely to be different because the random walks restart.

Similarly, domain-based similarity scores $s_{DO}(M_i, D_j)$ are computed and $s_{DO}(D_j, M_i)$ RWwR is used on the Domain-Device graph with the adapted transition matrix. $s_{DO}(M_i, D_j)$ and $s_{DO}(D_j, M_i)$ capture structural closeness between $M_i$ and $D_j$ in the Domain-Device graph.

### 4.2.3 A Running Example

An example Domain-Device graph is used to tell the difference between the similarity scores computed by RWwR and those computed by BAT-SU Zimmeck et al. (2017). Figure 4.2 shows the example graph. In the example, a user visits Twitter 500 times on a mobile device $M_1$ and visits Facebook 500 times on a desktop device $D_1$. Two other devices $M_2$ and $D_2$ that do not belong to the user visit both Facebook and Twitter 100 times.
The state-of-the-art BAT-SU essentially computes the similarity score of two devices using the common neighbors on a Domain-Device graph (or IP-Device graph). Specifically, on the Domain-Device graph, for each common domain visited by the two devices, BAT-SU computes a similarity score as the square root of the product of the normalized edge weights of the two corresponding edges. Then, BAT-SU adds such similarity scores for all common domains to get the final similarity score. In the example, BAT-SU would compute the similarity score between the user’s mobile device $M_1$ and desktop device $D_1$ as 0 since they do not have common neighbors. However, the similarity score between the mobile device $M_1$ and the desktop device $D_2$ is 0.71, which is larger than 0 as both devices visited Twitter. As a result, $M_1$ would mismatch $D_2$.

In RWwR, the similarity score between $M_1$ and $D_1$ are non-zero. Specifically, to compute the similarity between $M_1$ and $D_1$, we start a random walk from $M_1$. The particle will move to the Twitter node in the next time step. Then, the particle could continue moving to $M_2$, the node Facebook, and $D_1$. Therefore, when the random walk converges, $D_1$ would have non-zero stationary probability. Moreover, we calculate that $s_{DO}(M_1, D_1) = 0.06 > s_{DO}(M_1, D_2) = 0.055$, which indicates that $M_1$ correctly matches $D_1$. The reason is that RWwR can capture the latent relationships between Twitter and Facebook via using the data from other users, who visit both Facebook and Twitter frequently.

Note that such latent correlations do exist in practice. For instance, in the real-world dataset Zimmeck et al. (2017) used for evaluations in Section 5, domains are anonymized by hashing. It is found that user 49 did not visit domain A (hash value: b17e6ac8f4740bb465c6ac5cd0052b9bbc5ef49a0e58569eb00a87e2831ac5f) but visited domain B (hash value: 2da7c187647dc689d1686d250b24ae3b3551b24f88d90e58c2d49d5a3f1e617) 169 times on its mobile device. However, the user visited domain A 174 times but did not visit domain B on its desktop device. It is also found that 162 devices visited both domains, and they visited the two domains 363 and 862 times on average, respectively. Moreover, BAT-SU incorrectly matches user 49’s mobile device to user 1’s desktop device, while GraphTrack correctly matches the mobile and desktop devices for user 49. In fact, the existence of latent correlations is one of the key reasons why GraphTrack outperforms BAT.
4.2.4 GraphTrack using IPs or Domains Alone

Suppose we have \( m \) mobile devices \( M = \{M_1, \ldots, M_m\} \) and \( n \) desktop devices \( D = \{D_1, \ldots, D_n\} \). For each device, we have a list of IPs used by the device and domains visited by the device in its browsing history. Intuitively, if a mobile device matches a desktop device, then the desktop device is very likely to have the largest similarity score with the mobile device among the \( n \) desktop devices and the mobile device is very likely to also have the largest similarity score with the desktop device among the \( m \) mobile devices. GraphTrack leverages such symmetry between similarity to match mobile devices with desktop devices. Suppose we aim to match a mobile device \( M_i \). Next, IP-Device graph is used to illustrate the details of GraphTrack. Specifically, GraphTrack has the following three steps.

**Step I** Perform an RWwR from \( M_i \) in the IP-Device graph with the adapted transition matrix to compute similarity scores between \( M_i \) and each desktop device. Then, it is noticed that desktop device \( D_{IP} \) that has the largest IP-based similarity score with \( M_i \) among the \( n \) desktop devices. Formally,

\[
D_{IP} = \arg\max_{D_j \in D} s_{IP}(M_i, D_j).
\]

**Step II** Perform another RWwR from \( D_{IP} \) in the IP-Device graph with the adapted transition matrix to compute similarity scores between \( D_{IP} \) and each mobile device. Then, it is noticed that mobile device \( M_{IP} \) that has the largest IP-based similarity score with \( D_{IP} \) among the \( m \) mobile devices. Formally,

\[
M_{IP} = \arg\max_{M_j \in M} s_{IP}(M_j, D_{IP}).
\]

**Step III** If \( M_i = M_{IP} \), GraphTrack predicts that \( M_i \) and \( D_{IP} \) match, otherwise predicts \( M_i \) has no match. Similarly, domains alone can also be used to perform tracking. Denote the variants of GraphTrack that use IPs and domains alone as GraphTrack-IP and GraphTrack-Domain, respectively.

Note that the largest similarity scores in Step I and Step II could have ties. When such ties happen, all devices with the tied largest scores are considered and predict a match if any of them predicts a match. For instance, if a tie happens in Step I, Step II and Step III are applied for each tied device.
4.2.5 Combining IPs and Domains

We propose three methods to combine IPs and domains.

**GraphTrack-UniGraph** It integrates IPs, domains, and devices in a single unified graph, which is called *IP-Device-Domain graph*. Specifically, in the IP-Device-Domain graph, each unique IP, domain, mobile device, and desktop device is represented as a node; and an edge is created between an IP (or domain) and a device if the device used the IP (or visited the domain). Then this IP-Device-Domain graph is treated as if it was an IP-Device graph, and GraphTrack-IP is applied to perform mobile-desktop tracking. Specifically, for a mobile device $M_i$, an RWwR is performed from $M_i$ in the IP-Device-Domain graph to compute similarity scores between $M_i$ and each desktop device. Suppose the desktop device $D$ has the largest similarity score. Then, another RWwR is performed from $D$ in the IP-Device-Domain graph to compute similarity scores between $D$ and each mobile device $M_i$. Suppose the mobile device $M$ has the largest similarity score. If $M_i = M$, then predict that $D$ and $M_i$ match, otherwise predict that $M_i$ has no match. Denote this variant of GraphTrack as *GraphTrack-UniGraph*.

**GraphTrack-OR** It combines GraphTrack-IP and GraphTrack-Domain via the OR operator, i.e., predict a match if GraphTrack-IP or GraphTrack-Domain predicts a match. Denote this variant as *GraphTrack-OR*. Suppose we aim to match $M_i$. GraphTrack-IP produces an output $O_{IP}$, which is a desktop device or ⊥ (this means $M_i$ has no match using IP alone). GraphTrack-Domain produces an output $O_{DO}$, which is a desktop device or ⊥ (this means $M_i$ has no match using Domain alone). In GraphTrack-OR, if $O_{IP} \neq ⊥$, predict $M_i$ matches $O_{IP}$; else if $O_{DO} \neq ⊥$, predict $M_i$ matches $O_{DO}$; otherwise predict $M_i$ has no match.

**GraphTrack-AND** It combines GraphTrack-IP and GraphTrack-Domain via the AND operator, i.e., predict a match if both GraphTrack-IP and GraphTrack-Domain predict a match. Denote this variant as *GraphTrack-AND*. Specifically, if $O_{IP} \neq ⊥$, $O_{DO} \neq ⊥$, and $O_{IP} = O_{DO}$, predict that $M_i$ matches $D_{IP}$, otherwise predict that $M_i$ has no match. Compared to GraphTrack-OR, GraphTrack-AND is expected to predict less matches because it has a higher standard for a match.
4.3 Incorporating Manual Labels

Five unsupervised variants of GraphTrack are proposed, i.e., GraphTrack-IP, GraphTrack-Domain, GraphTrack-UniGraph, GraphTrack-OR, and GraphTrack-AND. We discuss how to adapt them to incorporate manual labels if they are available. A suffix “-SU” is appended to a method to indicate the supervised version, e.g., GraphTrack-IP-SU is the supervised version of GraphTrack-IP.

**GraphTrack-IP-SU, GraphTrack-Domain-SU, and GraphTrack-UniGraph-SU** Suppose we have $l$ manually labeled mobile-desktop pairs. Denote the pairs as $\mathcal{L} = \{(M_1, D_1), \cdots, (M_l, D_l)\}$, where the mobile device $M_i$ and the desktop device $D_i$ are known to match. Take GraphTrack-IP-SU as an example to illustrate how GraphTrack is adapted to incorporate manual labels. The key idea is to learn a *threshold* using the labeled pairs. Then, for an unlabeled mobile device, the similarity scores between the mobile device and each desktop device is computed using RWwR on the IP-Device graph. Suppose the desktop device $D_{IP}$ has the largest similarity score. If the largest similarity score is not less than the threshold, predict that the mobile device matches the desktop device $D_{IP}$, otherwise predict the mobile device has no match. Unlike GraphTrack-IP, the supervised GraphTrack-IP-SU does not rely on the similarity symmetry in a mobile-desktop pair, i.e., GraphTrack-IP-SU does not need Step II of GraphTrack-IP. This is because a threshold is learned using the labeled pairs and can use it to determine a match or not.

Next, we discuss how to learn the threshold. Roughly speaking, set the threshold as the smallest similarity score among the labeled mobile-desktop pairs. Specifically, a similarity score set $S = \emptyset$ is initialized. For each labeled pair $(M_i, D_i) \in \mathcal{L}$, perform an RWwR from $M_i$ in the IP-Device graph to compute similarity scores between $M_i$ and each desktop device. If the desktop device $D_i$ has the largest similarity score, which means that GraphTrack can correctly match $M_i$ with $D_i$ (can count a match even if other devices also have the largest similarity score), then add the similarity score between $M_i$ and $D_i$ into the set $S$. In the end, set the threshold to be the minimum similarity score in $S$. 
GraphTrack-Domain (GraphTrack-UniGraph) is adapted to GraphTrack-Domain-SU (GraphTrack-UniGraph-SU) in the same way, i.e., by replacing the similarity scores in GraphTrack-IP-SU as those computed using the Domain-Device graph and the IP-Device-Domain graph, respectively.

GraphTrack-OR-SU and GraphTrack-AND-SU: GraphTrack-OR-SU combines GraphTrack-IP-SU and GraphTrack-Domain-SU via the OR operator, while GraphTrack-AND-SU combines them via the AND operator. Specifically, suppose we aim to match $M_i$. GraphTrack-IP-SU produces an output $O_{IP}$, which is a desktop device or $\bot$; GraphTrack-Domain-SU produces an output $O_{DO}$. In GraphTrack-OR-SU, if $O_{IP} \neq \bot$, predict that $M_i$ matches $O_{IP}$; else if $O_{DO} \neq \bot$, predict that $M_i$ matches $O_{DO}$; otherwise predict that $M_i$ has no match. In GraphTrack-AND-SU, if $O_{IP} \neq \bot$, $O_{DO} \neq \bot$, and $O_{IP} = O_{DO}$, then predict $M_i$ matches with $O_{IP}$, otherwise predict $M_i$ has no match.

4.4 Computational Complexity

Complexity of unsupervised GraphTrack methods Denote the number of nodes of the IP-Device graph, Domain-Device graph, and IP-Device-Domain graph as $|V|_{IP}$, $|V|_{DO}$, and $|V|_{Uni}$, respectively. Moreover, denote the number of edges of the three graphs as $|E|_{IP}$, $|E|_{DO}$, and $|E|_{Uni}$, respectively. Note that $|E|_{Uni} = |E|_{IP} + |E|_{DO}$. For simplicity, take GraphTrack-IP as an example, since other unsupervised methods share the same analysis.

In Step I, GraphTrack-IP starts a RWwR from a mobile device and computes the adapted transition matrix $A$ in Equation 4.1 via traversing all edges of the IP-Device graph in $t_1$ iterations, where $t_1$ is the number of iterations that RWwR reaches the stationary distribution. Thus, the time complexity is $O(t_1 \cdot |E|_{IP})$. In Step II, it starts a RWwR from a desktop device and computes $A$ via traversing all edges of the IP-Device graph in $t_2$ iterations, where $t_2$ is the number of iterations that RWwR reaches the stationary distribution. Thus, the time complexity is $O(t_2 \cdot |E|_{IP})$. Usually, it is sufficient when $t_1 = t_2 = \log |V|_{IP}$ Tong et al. (2006). Step III is a simple prediction for each mobile device, and its time complexity can be ignored. Repeating above three steps for all $m$
mobile devices, we have the overall time complexity of GraphTrack-IP as $O(m \cdot \log |V|_IP \cdot |E|_IP)$, which is linear to $m$ and $|E|_IP$.

Likewise, time complexity of GraphTrack-Domain and GraphTrack-UniGraph are $O(m \cdot \log |V|_DO \cdot |E|_DO)$ and $O(m \cdot \log |V|_Uni \cdot |E|_Uni)$; and GraphTrack-OR and GraphTrack-AND have the same time complexity $O(m \cdot (\log |V|_DO \cdot |E|_DO + \log |V|_IP \cdot |E|_IP))$.

**Complexity of supervised GraphTrack methods** Denote the number of labeled mobile-desktop pairs for training as $m_{tr}$, and the number for testing as $m - m_{tr}$. Similarly, take GraphTrack-IP-SU as an example, since other supervised methods share the same analysis.

GraphTrack-IP-SU first uses $m_{tr}$ mobile-desktop pairs in the training set to learn a threshold. For each mobile-desktop pair, GraphTrack-IP-SU starts a RWwR from the mobile device, computes the stationary distribution of the RWwR among the IP-Device graph in $\log |V|_IP$ iterations, and calculates the similarity score with the paired desktop device. Repeating the process $m_{tr}$ times, GraphTrack-IP-SU can determine the threshold. Thus, the time complexity of training is $O(m_{tr} \cdot \log |V|_IP \cdot |E|_IP)$. Then, GraphTrack-IP-SU starts a RWwR from each mobile device in the testing set, computes the stationary distribution of the RWwR in $\log |V|_IP$ iterations, selects the desktop device with the highest similarity score, and compares the score with the threshold to determine whether there is a match. The time complexity for testing is $O((m - m_{tr}) \cdot \log |V|_IP \cdot |E|_IP)$. Thus, the time complexity of GraphTrack-IP-SU is $O((m - m_{tr}) \cdot \log |V|_IP \cdot |E|_IP)$, the same time complexity as GraphTrack-IP. Likewise, GraphTrack-Domain-SU, GraphTrack-UniGraph-SU, GraphTrack-OR-SU, and GraphTrack-AND-SU also have the same time complexities as their unsupervised versions.

**Complexity of the state-of-the-art methods** BAT uses the OR operator to combine IPs and domains, similar with GraphTrack-OR. For each mobile device, BAT respectively leverages IPs and domains to compute its similarity score with each desktop device via Bhattacharyya coefficient. Bhattacharyya coefficient is obtained by calculating the weighed common neighbors between the mobile device and each desktop device in the IP-Device graph and in the Domain-Device graph. In other words, this procedure is equivalent to traverse all edges of the IP-Device graph and the Domain-Device graph once. Therefore, the time complexity to match each mobile device
is $O(|E|_{IP} + |E|_{DO}) = O(|E|_{Unni})$. Repeating for all $m$ mobile devices, we thus have the overall
time complexity of BAT as $O(m \cdot |E|_{Unni})$. For BAT-SU, we have a similar analysis and its time
complexity is the same as BAT.

**Speeding up GraphTrack methods** Two-level parallel implementation can speed up Graph-
Track methods on large-scale datasets. First, different mobile devices can be run on different ma-
chines, as these mobile devices are matched in sequence. Second, each machine can parallelize
GraphTrack using multithreading. Specifically, nodes are firstly divided in a graph into groups.
Then in each iteration, each thread applies adapted RWwR in Equation 4.1 to a group of nodes
and the probability distributions obtained by all threads are combined to form the probability
distribution.
CHAPTER 5. EVALUATION

5.1 Experimental Setup

**Dataset description** A publicly available real-world dataset from Zimmeck et al. Zimmeck et al. (2017) is used to evaluate the proposed graph-based methods and compare them with state-of-the-art methods. Specifically, the dataset was collected at Columbia University and includes 126 users. 107 users have both a mobile device and a desktop device, and the remaining users have either a mobile device or a desktop device. The dataset includes the *IP addresses* used by each device and *Internet domains* visited by each device within around three weeks. Like previous study Zimmeck et al. (2017), we focus on the 107 users that have both a mobile device and a desktop device. The total number of unique IP addresses and Internet domains are 7,290 and 17,140, respectively. Moreover, each device used 85 unique IP addresses and visited 315 unique Internet domains on average. Note that the dataset also includes mobile apps used by each mobile device. However, such information has negligible impact on device tracking as shown by Zimmeck et al. Zimmeck et al. (2017). Therefore, such information will not be considered in this work for simplicity.

The used dataset has a small size, as it is fairly hard to collect a large dataset under an academic setting. However, a real-world cross-device tracking scenario could just involve a small number of devices as well. Specifically, a cross-device tracker could first narrow down millions of devices to a small number (e.g., hundreds) of candidates in a certain geographical region by analyzing their IP addresses. Then, the tracker analyzes the candidate devices. The intuition is that two devices that are geographically far away from each other are less likely to belong to the same user.

**Dataset preprocessing** The top-50 domains that have the most visits are filtered, because these 50 domains occupy around 70% of all visits, dominate the device similarities, and negatively impact the tracking performance. For instance, Figure 5.1 shows the performance of GraphTrack-
Domain with and without the top-50 domains. GraphTrack-Domain achieves much better performance when filtering the top-50 domains. Note that Zimmeck et al. Zimmeck et al. (2017) filtered the top-50 domains ranked by Alexa and all columbia.edu domains. As the released dataset anonymized each domain by cryptographic hashing, the same filtering cannot be applied.

**Evaluation metrics** Like previous study Zimmeck et al. (2017), *Accuracy, Precision, Recall, and F-Score* are used to evaluate cross-device tracking methods. These metrics involve True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). So these concepts are firstly defined. Suppose a method predicts some matched mobile-desktop pairs. TP is the number of pairs that truly match, and FP is the number of pairs that do not match in the ground truth. Suppose a method predicts that some mobile devices have no matched desktop devices. TN is the number of such mobile devices that truly have no matches, while FN is the number of such mobile devices that actually have matches. Given these terms, we have

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},
\]

\[
\text{Precision} = \frac{TP}{TP + FP},
\]

\[
\text{Recall} = \frac{TP}{TP + FN},
\]

and

\[
\text{F-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.
\]

**Compared methods** The methods are compared with the state-of-the-art supervised method proposed by Zimmeck et al. Zimmeck et al. (2017), which is denoted as *BAT-SU*.

**BAT-SU Zimmeck et al. (2017).**

It uses *Bhattacharyya coefficient* Wang and Pu (2013) to compute the similarity score between a mobile device and a desktop device. BAT-SU essentially computes the weighted common neighbors
between a mobile device and a desktop device in the IP-Device graph or the Domain-Device graph, where the weight is the normalized frequency of the corresponding IP or domain. We implement BAT-SU by ourselves and verify that the implementation achieves very close performance with the results reported by Zimmeck et al. (2017). More details will be discussed in Section 5.3.1.

**BAT.** BAT-SU is adapted to be an unsupervised method using GraphTrack framework. This unsupervised version is denoted as BAT. Recall that the key idea of unsupervised GraphTrack framework is that two devices are predicted to match only if their similarity is symmetric. Specifically, for a mobile device $M$, BAT-SU is used to compute the similarity score between the mobile device and every desktop device. Suppose the desktop device $D$ has the largest similarity score. Then, the similarity score between $D$ and each mobile device is computed, and the mobile device that has the largest similarity score is denoted as $M'$. If $M = M'$, then BAT predicts that $M$ and $D$ match, otherwise $M$ has no match. Moreover, BAT uses the OR operator to combine IPs and domains. In other words, BAT replaces the similarity scores in GraphTrack-OR as the Bhat-attacharyya coefficients used by BAT-SU. The OR operator is used because, as demonstrated later, it outperforms the AND operator and the unified graph at combining IPs and domains.

**Parameter setting** Set the restart probability $\alpha$ in a random walk to be 0.15 as previous work (Tong et al. 2006) since the value of $\alpha$ is found having a very small impact on the performance of GraphTrack. Moreover, the number of iterations is set to be $\log |V|$, where $|V|$ is the number of nodes in a graph (IP-Device, Domain-Device, or IP-Device-Domain graph). For all unsupervised methods, perform mobile-desktop tracking for each mobile device, i.e., for each mobile device in the dataset, either predict the matched desktop device or predict no match. For supervised methods, sample some mobile-desktop pairs as a training set and use the remaining as the testing set; use the training set to learn the thresholds and evaluate the methods using the testing set. When an experiment has randomness, repeat the experiment 5 times and report the mean and standard deviation. All experiments are conducted on a laptop with a 2.7GHz CPU and 8GB memory.
5.2 Results for Unsupervised Methods

In this part, unsupervised version of GraphTrack is evaluated, which is commonly used in the scenario of a pure third-party tracker, such as ads embedded in first-party domains. Because users do not interact with these trackers, e.g., logging into the tracker’s domain, such trackers do not have access to labeled device pairs.

**IPs are more informative than domains** Since mobile-desktop tracking is performed using IPs and domains, one natural question is that which type of data is more informative. The performance of GraphTrack-IP and GraphTrack-Domain are compared to answer this question. GraphTrack-IP (or GraphTrack-Domain) leverages IPs (or domains) alone to perform tracking. Figure 5.2 shows the results of GraphTrack-IP and GraphTrack-Domain at matching the mobile devices to desktops in the dataset. We observe that GraphTrack-IP performs slightly better than GraphTrack-Domain, which means that IPs are more informative than domains at tracking devices when using GraphTrack method. Two possible reasons are: 1) domains are more diverse than IPs on average in a device’s browsing history; 2) IP addresses somewhat indicate location information of users but different users may visit similar domains. Thus, IPs can better distinguish between devices of different users.

Note that previous studies Cao et al. (2015); Zimmeck et al. (2017) also found that IPs are more informative than domains at cross-device tracking, which is consistent with our observation.
Comparing different ways to combine IPs and domains

We discussed three ways to combine domains and IPs in Section 4.2.5. Accordingly, there are three variants, i.e., GraphTrack-UniGraph, GraphTrack-OR, and GraphTrack-AND. Figure 5.3 shows the comparison results. We observe that GraphTrack-OR consistently outperforms GraphTrack-UniGraph and GraphTrack-AND. GraphTrack-UniGraph combines IPs and domains using the IP-Device-Domain graph. This graph does not well distinguish the heterogeneous data types, i.e., IPs and domains. For instance, this graph does not distinguish between edges linking to IPs and edges linking to domains. As a result, GraphTrack-UniGraph achieves suboptimal performance. Intuitively, GraphTrack-AND should have a high Precision because it has a “higher” standard to predict a match for a mobile-desktop pair. Specifically, a mobile device and a desktop device are predicted to match only if both GraphTrack-IP and GraphTrack-Domain predict a match. This explains why GraphTrack-AND has a higher Precision than GraphTrack-UniGraph, though GraphTrack-AND has a much lower Accuracy, Recall, and F-Score. However, GraphTrack-AND is found achieves a lower Precision than GraphTrack-OR. The reason is, compared to GraphTrack-OR, GraphTrack-AND predicts a smaller number of both true matches (i.e., TPs) and false matches (i.e., FPs), but true matches decrease more than false matches. Thus, in the rest of this section, we focus on GraphTrack-OR.

Impact of mobile devices with no matches

There are 107 mobile-desktop pairs that are known to match in the dataset. In the real world, some mobile devices may have no matched
Figure 5.4: Impact of the fraction of unmatched mobile devices on the performance of GraphTrack-OR.

desktop devices in the dataset. For instance, a cross-device tracker (e.g., a certain Internet service provider) has collected the IPs and domains visited by a mobile device of a user, but does not have the IPs nor domains visited by desktop devices of the user, in which the user’s mobile device has no matched desktop device in the collected dataset. A natural question is how mobile devices with no matched desktop devices impact the performance of GraphTrack. To answer this question, \( x\% \) of mobile devices are randomly sampled from the 107 pairs, remove their matched desktop devices from the dataset, and then use GraphTrack-OR to perform tracking for the mobile devices in the dataset.

Figure 5.4 shows the results of GraphTrack-OR as \( x\% \) is increased from 0% to 50%. We observe that Accuracy fluctuates, while Precision, Recall, and F-Score tend to decrease, as more mobile devices have no matched desktop devices in the dataset. Precision and Recall decrease because less mobile devices that have matches are correctly predicted to match (i.e., TPs decrease) and more mobile devices that have no matches are predicted to have matches (i.e., FPs increase). F-Score decreases because Precision and Recall decrease.

**GraphTrack outperforms BAT** BAT-SU Zimmeck et al. (2017), state-of-the-art supervised method is adapted to an unsupervised method called BAT. Essentially, BAT is obtained by replacing the similarity scores in GraphTrack-OR as those computed by BAT-SU. Figure 5.5 compares GraphTrack-OR with BAT on the dataset with 107 mobile-desktop pairs. It is observed that...
Figure 5.5: Comparing GraphTrack-OR with BAT.

GraphTrack substantially and consistently outperforms BAT. The reason is that BAT predicts more mobile devices to have no matches, i.e., BAT has more False Negatives than GraphTrack-OR.

The results indicate that in the devices’ similarity scores computed by BAT, the *symmetry* property for the mobile-desktop pairs is weaker. More specifically, suppose $D$ is the desktop device that is the most similar to a mobile device $M$ among all the desktop devices; and $M'$ is the mobile device that is the most similar to the desktop device $D$. When the similarity scores are computed by BAT, $M' \neq M$ for more mobile devices, which means more False Negatives and explains BAT’s lower Accuracy, Recall, and F-Score. The fundamental reason is that BAT is essentially computing the common neighbors between a mobile device and a desktop device in the IP-Device graph and the Domain-Device graph, while GraphTrack leverages RWwR to analyze the complex graph structure.

**Robustness to uncertainty in single-device tracking** Single-device tracking (e.g., tracking via browser fingerprinting) could have uncertainty/errors, e.g., an IP or domain visited by one device may be linked to another device that did not access it. Therefore, one natural question is how uncertainty in single-device tracking impacts the performance of cross-device tracking. Since the dataset does not allow to implement a real-world single-device tracker, we simulate an inaccurate single-device tracker and study its impact on cross-device tracking. Specifically, for each web visit (characterized by an IP and a domain) from a device, the single-device tracker incorrectly links the visit to a randomly selected wrong device with a certain probability $y\%$. $y\%$ is called the *error*
Figure 5.6: Impact of error rates in single-device tracking on Accuracy of cross-device tracking.

rate of the single-device tracker. To simulate a single-device tracker with an error rate of $y\%$, randomly sample $y\%$ of the web visits of each mobile (or desktop) device and randomly distribute them to other mobile (or desktop) devices. Then use the noisy web visits of each device to perform cross-device tracking.

Figure 5.6 shows Accuracy of several mobile-desktop tracking methods as a function of error rates in single-device tracking. Recall that there are discussed several ways to deal with edge weights in the IP-Device graph and Domain-Device graph in Section 4. For instance, *GraphTrack-OR-UnWeighted* indicates do not use edge weights in GraphTrack-OR, i.e., set all edge weights to be 1. *GraphTrack-OR-UnNorm* indicates do not normalize edge weights by the weighted degree of devices, i.e., the weight of an edge between a device and an IP (or domain) is the number of times the device visited the IP (or domain).

There are several observations: First, GraphTrack-OR consistently outperforms BAT as the error rate increases. Second, GraphTrack-OR decreases slowly as the error rate increases, e.g., GraphTrack-OR’s Accuracy only decreases by 0.04 when the error rate is 10%. Third, GraphTrack-OR significantly outperforms GraphTrack-OR-UnWeighted and GraphTrack-OR-UnNorm. Specifically, when single-device tracking has no errors, GraphTrack-OR-UnWeighted has close performance with GraphTrack-OR. However, GraphTrack-OR-UnWeighted’s performance quickly decreases as the error rate increases. This is because as the error rate increases, the IP-Device graph and the
Domain-Device graph have much more edges that are noises, which significantly impact the graph structure based similarity scores. GraphTrack-OR-UnNorm is worse than GraphTrack-OR. This is because GraphTrack-OR-UnNorm is significantly influenced by devices’ weighted degrees, e.g., devices of heavy Internet users are biased to have large similarity scores. The results demonstrate that it is important to normalize edge weights of the IP-Device graph and the Domain-Device graph to increase both 1) performance when there are no errors in single-device tracking and 2) robustness to uncertainty in single-device tracking.

**Alternatives of converting BAT-SU to its unsupervised version** BAT-SU is proposed to convert to be unsupervised under our framework. Here, two other alternatives are explored to convert BAT-SU to be unsupervised. One way is to randomly select a threshold for BAT-SU without using a training dataset to learn it. Figure 5.7a shows the Accuracy of this unsupervised version of BAT-SU as the threshold is changed. The results indicate that BAT-SU highly depends on the threshold, and a threshold that is not carefully selected significantly degrades the performance. For instance, when the threshold is around 0.13, the Accuracy is the highest; when the threshold is 0.19, the Accuracy decreases by 21%. Interestingly, using the training dataset provided by the authors of BAT-SU, the learnt threshold is around 0.13, which explains the good performance of the supervised version. Moreover, the result indicates that the specific training dataset used by BAT-SU has already learnt the threshold that can achieve the best Accuracy.

Figure 5.7: Alternatives of converting BAT-SU to be unsupervised. (a) Accuracy vs. threshold. (b) Gaps between top two similarity scores for 20 sampled mobile devices.
For each mobile device, BAT-SU computes a similarity score with each desktop device. Therefore, another way could be to compute the gap between the largest similarity score and the second largest similarity score. If the gap is large enough (e.g., larger than a certain threshold), then BAT-SU predicts a match. Figure 5.7b shows such gaps for 20 sampled mobile devices. For each mobile device, the desktop device with the largest similarity score correctly matches the mobile device. It is observed that the gaps span a wide range of values, which means that it is challenging to select a gap threshold to achieve a good precision-recall tradeoff.

5.3 Results for Supervised Methods

In this part, the supervised version of GraphTrack is evaluated, which is commonly used in the scenario of a first-party tracker or a third-party one who also serves as the first party. In the former case like a bank website, a first-party tracker will only have access to labeled data from one single domain, i.e., the tracker domain; in the latter case like a Facebook Like button, the tracker will have access to a small number of labeled data if the user logs into the first-party tracker’s website. Both cases are evaluated below.

5.3.1 Verifying Implementation

Denote the implementation of BAT-SU by its authors Zimmeck et al. (2017) as BAT-Raw. Specifically, The same training set and testing set from the authors Zimmeck et al. (2017) are obtained. The training set consists of 63 matched mobile-desktop pairs and the remaining 44 matched pairs form the testing set. Table 5.1 shows Accuracy, Precision, Recall, and F-Score of BAT-Raw, BAT-SU, and GraphTrack-OR-SU. According to Zimmeck et al. (2017), BAT-Raw has 37 TPs, 5 FPs, 0 TN, and 2 FNs. The implemented BAT-SU has 36 TPs, 6 FPs, 0 TN, and 2 FNs. The results are not exactly the same as BAT-Raw because BAT-Raw filtered the top-50 domains ranked by Alexa and all columbia.edu domains which are anonymized by cryptographic hashing in the released dataset, while the top-50 most popular domains are filtered in the dataset instead.
Table 5.1: Results of BAT-Raw, the implemented BAT-SU, and GraphTrack-OR-SU on the same training set and testing set as in Zimmeck et al. (2017). BAT-Raw has 37 TPs, 5 FPs, 0 TN, and 2 FNs; BAT-SU has 36 TPs, 6 FPs, 0 TN, and 2 FNs; GraphTrack-OR-SU has 39 TPs, 2 FPs, 0 TN, and 3 FNs.

<table>
<thead>
<tr>
<th></th>
<th>BAT-Raw</th>
<th>BAT-SU</th>
<th>GraphTrack-OR-SU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.84</td>
<td>0.82</td>
<td>0.89</td>
</tr>
<tr>
<td>Precision</td>
<td>0.88</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>Recall</td>
<td>0.95</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>F-Score</td>
<td>0.91</td>
<td>0.90</td>
<td>0.94</td>
</tr>
</tbody>
</table>

5.3.2 Comparing GraphTrack with BAT and BAT-SU

For simplicity, GraphTrack-OR-SU is compared with the unsupervised version GraphTrack-OR and state-of-the-art method BAT-SU. Figure 5.8a shows the Accuracy of BAT, BAT-SU, GraphTrack-OR, and GraphTrack-OR-SU vs. labeled mobile-desktop pairs. When the fraction of training samples is $x\%$, $x\%$ of the 107 mobile-desktop pairs as labeled pairs are sampled and the sampled pairs are used to learn the thresholds in BAT-SU and GraphTrack-OR-SU; then the remaining mobile-desktop pairs are used as a testing set to evaluate them. For each $x\%$, repeat the experiments for 5 times, and compute the average and standard deviation of performance among the 5 trials.

**GraphTrack-OR-SU outperforms BAT-SU** From Figure 5.8a, we observe that GraphTrack-OR-SU significantly outperforms BAT-SU. For instance, when we have 20% of labeled mobile-desktop pairs, GraphTrack-OR-SU’s Accuracy is around 0.15 higher than BAT-SU’s. Moreover, BAT-SU requires labeling 40%-50% of mobile-desktop pairs in order to outperform unsupervised version GraphTrack-OR. The results indicate that modeling the interplays between IPs, domains, and devices using graphs is more powerful than common neighbors based similarity metrics used by BAT-SU at performing mobile-desktop tracking.

**Supervised methods outperform their unsupervised counterparts with enough training samples** Naturally, it is observed that supervised methods have better performance when more training samples are available. Specifically, both GraphTrack-OR-SU and BAT-SU have better performance as more training samples are available. Moreover, a supervised method requires a large fraction of labeled pairs to outperform its unsupervised counterpart. Specifically, GraphTrack-
Figure 5.8: Comparing GraphTrack with the state-of-the-art method, where (a) labeled device pairs are randomly sampled and (b) labeled device pairs are obtained via cross-device IDs on a single domain.

OR-SU outperforms GraphTrack-OR with 30% of labeled mobile-desktop pairs, while BAT-SU outperforms BAT with 30%-40% of labeled mobile-desktop pairs. The reason is that the matched mobile-desktop pairs have diverse patterns and supervised methods require labeling a large fraction of them as a representative training set in order to achieve good performance.

Labeled device pairs are obtained via cross-device IDs on a single domain A first-party tracker could obtain labeled device pairs via cross-device IDs. In particular, when a user logs in the tracker’s web service on both its desktop and mobile devices, the tracker can treat the user’s mobile-desktop device pair as a labeled device pair. To simulate such a first-party tracker, randomly select a domain in the dataset, treat it as the tracker’s domain, take the mobile-desktop pairs that both visited the domain as labeled device pairs, and use the remaining devices to evaluate GraphTrack-OR-SU and BAT-SU. Figure 5.8b shows the comparison results, where repeat the experiments five times on five randomly selected domains and report the average results. It is observed that GraphTrack-OR-SU also substantially outperforms BAT-SU.

Scalability The scalability of GraphTrack and BAT are evaluated in terms of their running time. Since it is necessary to vary the number of mobile devices in the IP-Device graph, Domain-Device graph, and IP-Domain-Device graph, scalability on synthesized graphs is evaluated with different number of nodes. Note that the number of edges in the graphs also changes when the
number of nodes varies. In synthesized graphs, assume that the number of mobile devices and
desktop devices are the same. Moreover, since in the real-world dataset used for evaluations, each
device used 85 unique IPs and visited 315 unique domains on average, each node has an averaged
85 edges on the synthesized IP-Device graphs, 315 edges on the synthesized Domain-Device graphs,
and 400 edges on the synthesized IP-Domain-Device graphs. That is, $|E|_{IP} = 85m$, $|E|_{DO} = 315m$,
and $|E|_{Uni} = |E|_{IP} + |E|_{DO} = 400m$. Then, the dominant time complexities for both unsupervised
and supervised GraphTrack methods are $O(m^2)$, with different methods having different constant
factors.

Figure 5.9: (a) Running time of unsupervised GraphTrack methods. (b) Running time of supervised methods.

Figure 5.9a and 5.9b respectively show the running time of unsupervised GraphTrack methods
and supervised GraphTrack methods, as well as BAT and BAT-SU, in a log-log scale. First, the
running time of all unsupervised methods and supervised methods are quadratic with respect to
the number of mobile devices (the slopes of the lines in the log-log scale are close to 2). Second,
supervised methods are more efficient than their unsupervised versions. This is because unsupervised
methods use a symmetric similarity to predict match, while supervised methods do not rely
on the similarity symmetry and do not need the Step II of unsupervised methods. Third, without
parallelization, BAT and BAT-SU are more efficient than unsupervised GraphTrack methods and
supervised GraphTrack methods, respectively. This is because BAT and BAT-SU traverse all edges
in a graph once, while GraphTrack methods involve iteratively traversing all edges until convergence. Fourth, with parallelization using multi-threading, unsupervised GraphTrack methods and supervised GraphTrack methods are comparable with BAT and BAT-SU. For simplicity, only the running time of paralleled GraphTrack-OR and paralleled GraphTrack-OR-SU using 16 threads are shown, and are denoted as GraphTrack-OR-Parall and GraphTrack-OR-SU-Parall, respectively.
CHAPTER 6. SUMMARY AND DISCUSSION

Cross-device tracking when users have multiple mobile devices or multiple desktop devices

The case that is mainly discussed is that each user has one mobile device and one desktop device. However, GraphTrack is also applicable to the following cases: 1) a user has multiple mobile devices and one desktop device; and 2) a user has one mobile device and multiple desktop devices. In case 1), GraphTrack can be applied to each mobile device. In case 2), GraphTrack can be used to perform desktop-mobile tracking and can be applied to each desktop device. However, GraphTrack is not suitable for users having both multiple desktop devices and multiple mobile devices. Since for each mobile device, GraphTrack selects only one desktop device having the largest similarity score with the mobile device.

Handling other cross-device tracking problems

The mainly discussed case is mobile-desktop tracking as it is empirically observed in the wild. However, GraphTrack can be easily extended to desktop-mobile tracking, mobile-mobile tracking, and desktop-desktop tracking. The only difference is either starting a RWwR from each desktop device, e.g., in desktop-mobile and desktop-desktop tracking, or constructing a different IP-Device graph and Domain-Device graph, e.g., in mobile-mobile and desktop-desktop tracking.

Defense against GraphTrack

GraphTrack has shown the robustness to uncertainty in single-device tracking when incorrectly linking a fraction of IPs and domains to randomly selected wrong devices. To defend against GraphTrack, it is necessary to design a more intelligent strategy. One potential direction is to leverage adversarial machine learning techniques. For instance, we can convert the problem of redistributing links between IPs, domains, and devices into an optimization problem, where the objective function is to minimize the GraphTrack performance and the constraint restricts the fraction of incorrect links. Then, the solution to the optimization problem could be specially designed incorrect links that minimize the performance of GraphTrack.
Limitation in the usage of IP address As acknowledged in Zimmeck et al. (2017), even if two devices have the same IP address, they may not belong to the same user, especially when there are more than one user in a household. In this perspective, GraphTrack has the same limitation as BAT.

Limitation of the dataset The used dataset Zimmeck et al. (2017) has a small size. However, as discussed in Section 5.1, a real-world cross-device tracking scenario could only involve a small number of devices. Moreover, in the dataset, each user only has one mobile device and one desktop device. It would be a valuable future work to apply GraphTrack to handle other cross-device tracking scenarios.
REFERENCES


Cao, Y., Li, S., and Wijmans, E. (2017b). (cross-)browser fingerprinting via os and hardware level features. In *NDSS*.


Perry, M., Clark, E., and Murdoch, S. (2015). The design and implementation of the tor browser [draft][online], united states.


Bruner (1960)