UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Applied ML for Robust Network Applications

Permalink

https://escholarship.org/uc/item/7s32x3ck

Author

Fahim, Abdelrahman

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Applied ML for Robust Network Applications

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Abdulrahman Fahim

September 2024

Dissertation Committee:

Dr. Srikanth V. Krishnamurthy, Chairperson
Dr. Zhiyun Qian
Dr. Evangelos Papalexakis
Dr. Zhaowei Tan

Copyright by
Abdulrahman Fahim
2024

The Dissertation of Abdulrahman Fahim is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I want to express my gratitude to the University of California, Riverside, and the Department of Computer Science and Engineering for the excellent facilities and resources provided throughout my PhD journey.

I am deeply thankful to my advisor, Professor Srikanth Krishnamurthy, for his invaluable guidance, unwavering support, and patience throughout my studies. His extensive knowledge has been a source of inspiration and encouragement in both my academic research and daily life.

I also extend my thanks to Professor Vagelis Papalexakis and Professor Zhiyun Qian for their relentless efforts in mentoring me and helping me grow as a scientist.

My appreciation goes to my friends—Basem, Refaat, Ahmed Abdo, Sedqy, Foad and Charle—for the cherished moments we've shared together. A special thank you to my wife, Nouran, who has been a constant source of support through all the challenges, my absences, and moments of frustration. Her encouragement, idea exchanges, and support in managing our family during my graduate studies have been invaluable.

I am deeply grateful to my father, Professor Yousef Fahim, and my mother, Professor Nahed Abdellatif, for their unwavering love and for providing me with the opportunities and experiences that have shaped who I am today.

To my child, Dan, you are the most cherished and meaningful part of my journey through graduate studies.

I also want to thank my family back home for their constant and exceptional love, assistance, and support. I am especially thankful to my sisters—Esraa and Rawda, for always being there for me.

Lastly, I honor the memory of my beloved aunts, Nothaila and Faiza, my grandmothers, Zahira and Zebeda, and my uncle, Abdo, who passed away during my graduate studies. I regret not having had the chance to say goodbye.

ABSTRACT OF THE DISSERTATION

Applied ML for Robust Network Applications

by

Abdulrahman Fahim

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2024
Dr. Srikanth V. Krishnamurthy, Chairperson

Machine learning has opened up numerous opportunities and applications for various networking applications. This dissertation focuses on ML-based solutions in constrained environments for different networking applications.

First, we present a framework that detects and summarizes key global events from distributed crowd-sensed data in a bandwidth-constrained environment. We introduce BigEye, a novel framework that only transfers limited data from distributed producers to a central summarizer, supporting highly accurate detection and concise visual summarization of key events of global interest.

Second, we develop AcTrak, a framework to control steerable cameras through a network to retrieve telemetrics of interest. AcTrak automates a camera's motion to switch appropriately between zooming in on existing targets in a scene to track their activities and zooming out to search for new targets arriving in the area of interest. We aim to achieve a good trade-off between these two tasks, ensuring that new targets are observed by the camera before they leave the scene while frequently monitoring the activities of existing

targets.

Third, we uncover a vulnerability that enables fast and stealthy data exfiltration over DNS channels. While existing defenses against such attacks appear robust, we demonstrate that our carefully designed and novel DNS exfiltration attack, Dolos, which uses a generative adversarial network (GAN), can encode sensitive data to evade these detectors unlike existing state-of-the-art attack methods. Additionally, Dolos can significantly expedite exfiltration compared to prior methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an era where digital technology and data play crucial roles in various domains, the integration of advanced systems for real-time monitoring, data analysis, and security is becoming increasingly significant. This thesis explores the machine learning based solutions in constrained environments for different networking applications. This thesis explores innovations in three key areas: distributed real-time event detection and summarization in network constrained environment, dynamic control of Pan-Tilt-Zoom (PTZ) surveillance cameras over a network to retrieve telemetrics of interest, and stealthy data exfiltration over DNS channels.

Distributed real-time event detection and summarization: Social media platforms have emerged as critical sources of real-time information, particularly in disaster scenarios and significant events. Posts shared by individuals on platforms such as Twitter serve as a form of crowd-sensing, where user-generated content acts as an informal yet potent sensor network. The challenge lies in efficiently managing and processing this vast

amount of data to detect global events with minimal latency. Prior work has demonstrated that raw data from geographically dispersed sources can overwhelm central processing systems and networks, necessitating solutions that focus on efficient data summarization and event detection. This thesis builds upon these challenges by introducing `BigEye`, a system designed to detect key global events with high accuracy while significantly reducing the volume of transmitted data. `BigEye`'s approach involves transferring minimal metadata to a central entity for event detection and subsequently retrieving only essential visual content, thereby addressing the issues of data overload and network congestion. We demonstrate that `BigEye` achieves equivalent accuracy in detecting key events as a system with centralized data access, despite transferring only 1% of the raw data volume. Additionally, `BigEye`'s parallelized approach to transferring visual content results in a 67% reduction in average delay compared to baseline methods. Our solution as well as the results are detailed in Chapter 2 of this dissertation.

Dynamic Control of PTZ Surveillance Cameras: As surveillance technology becomes more prevalent, there is a growing need for sophisticated camera systems that provide comprehensive scene coverage while capturing detailed information on specific targets. Traditional fixed cameras often fall short in dynamic environments, where both broad scene coverage and fine-grained activity monitoring are essential. Pan-Tilt-Zoom (PTZ) cameras offer a solution with their ability to adjust their field of view and zoom capabilities. This thesis presents a novel framework that optimally controls PTZ cameras to balance the dual objectives of quickly identifying new targets and capturing high-resolution images of existing ones. By employing a Markov Decision Process (MDP) and reinforcement learning

techniques, Our method adapts to the evolving scene to manage the trade-offs between coverage and detail, thus enhancing the effectiveness of surveillance systems. Through simulations with real datasets, we demonstrate that AcTrak identifies newly arriving targets 30% faster than a non-adaptive baseline and has a significantly lower miss rate, detecting nearly all targets compared to the baseline, which can miss up to 5%. Additionally, our implementation of AcTrak on a real camera shows that it captures approximately twice as many high-resolution images of targets as the baseline. The details of this problem formulation and the underpinning algorithms for dynamic control of PTZ cameras is described in details in Chapter 3.

Stealthy and fast DNS exfiltration attachs: In the domain of cybersecurity, data exfiltration remains a significant threat, particularly when attackers use sophisticated techniques to covertly extract sensitive information. One such method is DNS exfiltration, where data is hidden within DNS queries to evade traditional security measures. Despite advances in detection technologies, attackers continually refine their methods to bypass these defenses. This thesis introduces DOLOS, a cutting-edge DNS exfiltration tool designed to circumvent modern defenses through a generative adversarial network (GAN)-based encoding-decoding framework. DOLOS employs a generative model to create DNS queries that are nearly indistinguishable from benign traffic, effectively evading state-of-the-art detection systems. By optimizing the encoding process and incorporating a novel rate-tuning module, DOLOS achieves high efficiency and stealth in data exfiltration. The approach not only demonstrates resilience against current detection techniques but also provides insights into the ongoing evolution of exfiltration methods. Our evaluations reveal that Dolos maintains

a detection probability of 12%, even when six of the nine state-of-the-art defenses we tested are used simultaneously. In contrast, achieving the same detection rate with current baseline exfiltration techniques would almost certainly lead to detection. If baseline techniques reduce their rates to achieve a detection probability slightly higher than Dolos (0.15), they take 25 times longer to complete the exfiltration. With the remaining three defenses, the baselines are almost invariably detected, whereas Dolos remains relatively undetectable regardless of the exfiltration rate. The detailed DOLOS 's implementation and evaluations can be found in details in Chapter 4.

Chapter 2

BigEye: Detection and Summarization of Key Global Events from Distributed Crowd-sensed Data

Social media postings using smartphones (referred to as crowd-sensed data) can often facilitate real time detection of key physical events in applications like disaster recovery or in smart cities. These postings also often contain visual content (e.g., images) that can be used to obtain zoomed-in views of such events. This crowd-sensed data is likely to be of large volume and distributed across a plurality of producers (e.g. cloudlets). Blindly transferring this large volume of raw data from the producers to a consumer will induce information overload and consume very high bandwidth. The problem is exacerbated in

scenarios with limited bandwidth (e.g., after a disaster).

In this work, we design `BigEye`, a novel framework that only transfers very limited data from the distributed producers to a central summarizer, and yet supports (a) highly accurate detection and (b) concise visual summarization of key events of global interest. In realizing `BigEye`, we address several challenges including (a) identifying events that have the highest global interest via the transfer of appropriate limited metadata from the producers to the summarizer (b) reconciling metadata that could be inconsistent across the producers and (c) timely retrieval of visual summaries of the key events given bandwidth constraints. We show that `BigEye` achieves the same accuracy in detecting key events, as a system where all data is available centrally while transferring only 1 % of the raw data volume. Compared to baseline approaches, `BigEye` 's parallelized transfer of visual content reduces the average delay by 67 %.

2.1 Introduction

People share and disseminate postings on real-life events on social networks via smartphones (which can be considered the de facto most widely used IoT devices today). Such postings provide an inherent sensing capability [255] (defined as crowd-sensing or social sensing in the literature [97]). Specifically, as pointed out in prior papers [16, 255, 259], user posts relating to real-time information about events (e.g., protests, earthquakes) on social media networks, can be considered as IoT sensor outputs that provide knowledge of interest with regards those events. In fact, news reports suggest that Harvey storm victims used social media to communicate about critical events [202] during/after the disaster. We also

point out that it is quite common for such user postings (e.g., using Twitter) to contain images that can be used for composing *zoomed-in* (or more informative) visual summaries of key events.

In scenarios like disaster aftermaths, one can visualize crowd-sensed data to be dispersed across a set of geographically distributed "producers," because of strained infrastructure. For example, one can envision the user postings (we also call them microblogs) within a local geographical region to be sent (streamed) to a common server (e.g., a cloudlet [213] or a local server [136, 237]), which can be considered to be a *producer*. While using microblogs as IoT sensed outputs towards detecting physical events has received some recent attention (e.g., [16, 259]), these prior efforts assume that all crowd-sensed data is available centrally instead of being distributed across geographically spread producers; the latter is a more accurate representation of what happens in practice.

Blindly transporting all the raw data from such producers to either the consumer, or to a central entity for analytics will not only cause an information overload at that entity, but also strain the network in terms of bandwidth consumption potentially resulting in network congestion. In addition, a large part of this voluminous data will have redundant information, and might not contribute to extracting useful information (e.g., detecting events of interest). The problem of data transportation is exacerbated in scenarios such as natural disasters wherein it is very common for network infrastructure to be damaged; in such cases, the available bandwidth is constrained significantly making it prohibitive to transfer raw data. Again, going back to the example of the tropical storm Harvey, the failure of several cell towers [236] strained the available bandwidth. Because of this, it becomes

imperative to only retrieve small amounts of data from the distributed producers, and yet be able to detect events of global significance with high accuracy. Our work targets this important problem.

Specifically, in this paper, we design an IoT service, we call `BigEye`, using which global events can be detected with very high accuracy, by only sharing very small amounts of information (metadata) between the distributed producers (with raw crowd-sensed data) and a central entity (which we call summarizer). Once global events are detected, `BigEye` facilitates the transport of limited content in the form of images/videos to the summarizer, which then composes a visual summary from the same detected events to provide the appropriate information to gain insights with regards to these events of interest. These summaries can then be sent to a consumer (e.g., search and rescue personnel after a disaster). We implement `BigEye` using Twitter (which provides user postings of the type of interest) as a proxy, and showcase its benefits. Thus, without loss of generality, in the rest of the paper we refer to user postings as tweets.

Challenges: The challenges in designing `BigEye` are multifold. First, each producer only has a local view of events and thus, is unable to by itself determine which events are of global significance. On the other hand, transferring all local data from all producers to the central summarizer is prohibitive and wasteful. The challenge then is "how do we identify key global events by only transferring limited metadata from each producer to the summarizer?" Second, multiple producers may detect the same global event via sensed triggers, but there is no easy way to determine that the triggers correspond to the same event. Unless we are able to make this determination, redundant content (e.g., unnecessary

visual data) may be retrieved by multiple producers. Thus, "how do we reconcile (possibly inconsistent) metadata from multiple producers that correspond to a common event?" Finally, once the events are detected, we seek to retrieve a limited amount of visual content with minimum latency from the multiple producers to compose a summary. The challenge here is to identify "which producer will send what content in order to maximize the level of parallelization of retrieval, given that the bandwidths to the different producers could be different."

Approach in brief: Briefly, with `BigEye`, each producer first, individually identifies local events likely to have global importance (or interest) using a metric which is called the local information gain. The producer then *pushes* metadata pertaining to these local events to a central summarizer. The summarizer then assesses the need for any additional metadata and *pulls* the same from the proper subset of producers. As one of our contributions, we formally show that via an appropriate choice of the number of local events at each producer, we can guarantee that the detected key global events are identical to the ones detected if all the data is made available centrally. Once the global events are detected, `BigEye` uses a lightweight method to reconcile common events across a plurality of producers. Finally, using lightweight measurements of bandwidths to the various producers, it uses an intelligent algorithm to parallelize the transfer of visual content from the producers to compose visual summaries of all such events within a very short time (ideally we want to do this in minimum time but we show this is NP-hard). This paper is an extension of our work [74], where in this work we introduce all `BigEye` modules and show its overall merits.

Summary of our Contributions. A summary of our contributions in this paper are as follows:

- We propose a method within `BigEye` that allows the detection of key global events when the crowd-sensed data is distributed across multiple distributed producers. Via the transfer of just 1 % of the data from the producers to a central summarizer (in the form of metadata), we show that `BigEye` is able to detect all key global events that would have been detected if all the data was available centrally.
- `BigEye` includes a module that reconciles events across producers without having to transfer the entire content corresponding to these events to the central entity (§ 2.5). We call this "consolidation" of events across producers. We show that consolidation further reduces the communication costs by 60% on average.
- We show that we can map the problem of selecting visual objects with the highest scores to be sent from each producer to the summarizer, such that we ensure a certain number of visual objects per event while minimizing latency, to a multi-dimensional knapsack instance [100]; this is an NP-Hard problem. Given this, we design an online algorithm that (a) dynamically estimates bandwidth to each producer, and (b) fetches visual objects in parallel greedily from the producers given these estimates. Our experiments show that compared to baseline approaches, `BigEye`'s parallelized transfer of visual content reduces delay by 67 %.

2.2 Baseline case: Centralized detection

We first provide a description of a baseline method which allows global event detection when all the data are centrally available. The approach is largely based on the approach (called Storyline) by Wang et al., in [259]. We start by describing the "information gain metric" used to detect global events in Storyline in this section. In § 2.4, we show how we can transition from this global metric to an appropriate local metric that facilitates the detection of key global events in the distributed settings that we are interested in.

"Information gain" is a commonly used metric for detecting discriminative features [270]. In Storyline, this metric is used to measure the burstiness in the co-occurrence of pairs of uncommon words (keyword pair) in a stream of tweets (microblog objects are used as sensor outputs) between two time epochs. The keyword pairs are considered to be *discriminative features* and to be associated with physical events of interest. For example, given a key physical event where "a drunk driver kills a running dog on the bridge," the microblog data would experience a bursty surge in the keyword pair (driver, drunk) at a specific time epoch. The approach enables event demultiplexing, i.e., identifying separate events that belong to the same global scope. For example, in a disaster scenario one might be interested in distinguishing between multiple areas with humans in distress (and not identify them as the same issue) so that the human responders can take proper actions. Formally, the information gain associated with a keyword pair s_z, across time windows $k-1$ and k is given by infoGain [259]:

$$infoGain = H(Y) - H(Y|s_z) \tag{2.1}$$

In the above equation $H(Y)$ and $H(Y|s_z)$ are computed as follows:

$$H(Y) = -\frac{N_k}{N_k + N_{k-1}} \log \frac{N_k}{N_k + N_{k-1}} - \frac{N_{k-1}}{N_{k-1} + N_k} \log \frac{N_{k-1}}{N_k + N_{k-1}} \text{ and,} \tag{2.2}$$

$$H(Y|s_z) = -\frac{N_k^z}{N_k^z + N_{k-1}^z} \log \frac{N_k^z}{N_k^z + N_{k-1}^z} - \frac{N_{k-1}^z}{N_{k-1}^z + N_k^z} \log \frac{N_{k-1}^z}{N_k^z + N_{k-1}^z} \tag{2.3}$$

where, N_k is the number of tweets in the current (k^{th}) time window, and N_{k-1} is the number of tweets in the immediately previous time window, $k-1$. N_k^z and N_{k-1}^z are the number of tweets that contain the keyword pair s_z in the current and previous time window, respectively. Note that roughly these expressions characterize the entropy (conditional entropy) relating to the occurrence of tweet volume (number of tweets with the specific keyword pair) in consecutive time windows. More details can be found in [259] and are omitted in the interest of space. A threshold is chosen, and all keyword pairs whose information gains are higher than that threshold are considered to have associated physical events that are of interest.

2.3 Overview and assumptions

We envision that `BigEye` is used in an architecture that comprises a set of producers, a summarizer and one or more consumers (users). `BigEye` seeks to identify key events of global interest, occurring at specific time windows of fixed duration (we also call these windows epochs). A producer is an entity that collects sensed data from within a local region (e.g., a microblog repository). Let m be the number of producers; each producer is denoted

by P_i, where $i = 1, 2, ..., m$. In `BigEye`, we assume that the time epochs are synchronized across all producers (we assume that protocols such as NTP can enable this [172]); without loss of generality, we denote the index of the time window of interest by k.

The summarizer is a central entity (e.g., a server) that receives appropriate data from all the producers, identifies key events, and composes a summary. We assume that all the producers are connected to the summarizer via a network of arbitrary topology. The transfer of all local data from the m producers to the central summarizer is considered to be large and prohibitive in terms of bandwidth consumption (either because of limited bandwidth, congestion or both). This will typically be the case in scenarios such as disaster recovery, wherein loss of infrastructure can induce significant constraints on the available bandwidth (e.g., fewer functional base stations or access points). Blindly sending the large volume of data can have significant consequences. First, because of the sheer volume, significant delays may be experienced. This in turn induces significant delay in aggregating the data of interest (or importance) from the multiple producers. This in turn, delays the inferences relating to events detection, which can cause response delays (especially in disaster scenarios when fast response is critically). We point out that if instead, inferences are based on partially received data, sub-optimal decisions may be made and the response resources may be directed at less than critical points of need.

Without fine-grained information relating to the events of interest, the summarizer may experience confusion with respect to multiple events, often categorizing them to be the same. This increases the rate of false negatives (some of these may be missed completely). It is also possible, that the false positive rates relating to events could increase due to decisions

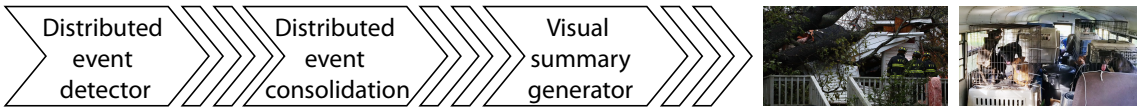

Figure 2.1: A high level depiction of `BigEye` with its modules

made with coarse grained information. In this work, we propose, `BigEye`, a system that deals with the aforementioned conditions and is able to detect events accurately with low latency. Furthermore, it has the ability to disambiguate events that occur across a wide reach (referred to as global events), thereby allowing their precise identification for proper response delivery.

In `BigEye`, each producer identifies a set of local events that are likely to be of global interest or importance. It generates appropriate metadata corresponding to these events, and pushes the metadata to the summarizer. While multiple producers may report the same global event albeit with inconsistent metadata, `BigEye` allows the summarizer to reconcile/consolidate such events. With `BigEye`, the summarizer also triggers the intelligent retrieval of visual content from the appropriate producers to compose visual summaries of all the key events within a very short time. These precise visual summaries will enable disambiguation across events, thereby allowing response delivery to all events that are of importance. A depiction of the modules and composition of `BigEye` is shown in Fig. 3.3.

A consumer or user is connected to the summarizer, and queries for summaries of key events. We assume that a query provided by a user belongs to a specific scope (e.g., protest, army conflicts etc.) and the producers collect the data that match such

queries[1]. Each producer observes postings composed of short descriptions of their context with attached multimedia (videos/images) (e.g., tweets). `BigEye` takes this information as an input and produces summaries that are composed of global events of interest plus a visual summary that provides additional context which provide detailed information and also allow disambiguation of similar but separate events.

Motivating our vision: To exemplify our vision, we describe an example where our framework can find application and help significantly. A rescue operator is interested in identifying/detecting the key global events in a disaster affected area. Let us recall the previously mentioned example relating to hurricane Harvey, wherein the victims posted tweets relating to their surroundings, with textual and visual content. The postings are sent to the nearest cell tower or cloudlet (we refer to these as producers in this paper); because of bandwidth constraints, carrying a possibly a large sets of tweets with visual content beyond this local infrastructure in near real-time will be problematic. Thus, a central controller where the rescue operator (user) resides might not be able to receive this information, or process it in a timely manner. Here is where `BigEye` comes into place; it enables the rescue operator who only has limited connectivity to the producers, to identify the key global events without retrieving the raw generated data in its entirety, from the producers. For example, the global event may be a "major flooding in a specific street or neighborhood", "pets are in danger in a certain area" or "the 911 network has broken-down". The rescue operator can then take proper action, based on on the detected "event level" information.

The key notations used in the paper are summarized in Table 3.1.

[1]The communications between the user and the summarizer are out of the scope of this paper.

2.4 Distributed Event Detection

Compared to prior approaches (in particular the baseline described in § 2.2), `BigEye` distinguishes itself in that it applies in a realistic scenario where the crowd-sensed data is dispersed across a plurality of geographically distributed producers. If one were to blindly transporting all the raw data that is available at these producers to a central summarizer (which can then create summaries as discussed before), the bandwidth consumed will be prohibitive, especially in scenarios such as disaster recovery. Because of this constraint, when building `BigEye` we try to answer the question "How can we determine the events that are of global significance without having to transfer all raw data to the central summarizer?"

The information gain metric previously discussed in § 2.2 is based on an underlying assumption that the entire raw data is centrally available. However, this assumption does not hold and each producer only has a local view. Thus, needed is an approach wherein the significance of a local event can be estimated with respect to its the global importance (i.e., will a local event also be flagged as a key global event if the data was available centrally ?). If all the producers were to simply report the "number of tweets" to the summarizer, in the two consecutive time windows of interest (say $k-1$ and k), $H(Y)$ can be computed. However, the challenge lies in computing $H(Y|s_z)$ globally since (a) all keyword pairs and the associated tweets are not known centrally and (b) each producer will only see part of the data and can only compute $H(Y|s_z)$ based on its local dataset. To address these issues, we first map a global requirement on information gain (and thus $H(Y|s_z)$) to a local requirement at each individual producer. Later, we reconcile inconsistencies by having the

summarizer pull appropriate data from a subset of producers.

What values should the global $H(Y|s_z)$ take to achieve high information gain? Before, we describe our approach in more detail, we first ask the above question. As pointed out above, $H(Y)$ only depends on the number of tweets in consecutive time windows. Thus, the discriminatory term that dictates the information gain with respect to a keyword pair (say s_z) is $H(Y|s_z)$. It is obvious that the lower the value of this term, the higher the information gain associated with the keyword. With reference to Equation 2.3, let us define $p_s = \frac{N_k^z}{N_k^z + N_{k-1}^z}$. Then, $\frac{N_{k-1}^z}{N_{k-1}^z + N_k^z} = 1 - p_s$. In Fig. 2.2, we plot $H(Y|s_z)$ as a function of p_s. We see that the lowest values of $H(Y|s_z)$ (yielding the highest information gain) are achieved when p_s is very small or very large (approaches 1). Since p_s corresponds to the probability of having a high number of tweets in frame k relative to the previous frame, it must be large (not small) in order to reflect a new event of interest (otherwise it indicates an event that was of interest in frame $k-1$ but has died down). In other words, the takeaways from the above discussion are (a) $H(Y|s_z)$ must be small (say some small value ϵ) and (b) the corresponding probability p_s as defined above must be large (we require it to be > 0.5).

Let r be *the ratio of occurrence of a keyword pair in the current time epoch to the corresponding occurrence in the previous time epoch.* Let p_s^* be the value of p_s that makes $H(Y|s_z) = \epsilon$. Since we cannot directly obtain p_s^* in closed form by solving $H(Y|s_z) = \epsilon$, we numerically solve it using the Newton method [159] and from among the results, choose the value that is > 0.5. From p_s^*, we compute $\frac{N_k^z}{N_{k-1}^z}$ and denote it as r^*. r^* is the minimum (global) threshold with respect to the ratio of occurrences of a keyword pair in consecutive

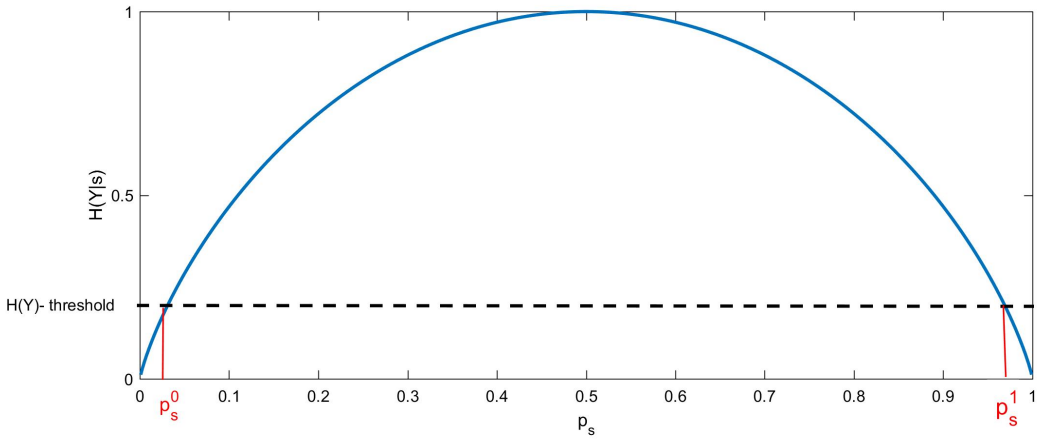

Figure 2.2: $H(Y|s)$ with varying p_s. p_s^0 and p_s^1 are two intersecting points with $H(Y) - threshold$

time epochs, that must be met if the associated keyword pair is to signify an event of interest. In other words, if for a keyword pair $r \geq r^*$, then that keyword pair is a discriminative pair. Next, we provide a formal proof of this claim.

Lemma 1 *Any pair with $p_s \geq p_s^*$ has a ratio of occurrence, r greater than or equal to r^**

Proof.

$$p_s = \frac{N_k^z}{N_k^z + N_{k-1}^z} \tag{2.4}$$

$$p_s N_k^z - N_k^z = -N_{k-1}^z p_s \tag{2.5}$$

$$\frac{N_k^z}{N_{k-1}^z} = \frac{p_s}{1 - p_s} \tag{2.6}$$

Note that r is nothing but $\frac{N_k^z}{N_{k-1}^z}$. If we can show that $\frac{p_s}{1-p_s}$ is non decreasing function i.e., $\frac{p_s}{1-p_s} \geq \frac{p_s^*}{1-p_s^*}$ if $p_s \geq p_s^*$, then we can infer that $r \geq r^*$ if $p \geq p_s^*$. Let us denote $\frac{p_s}{1-p_s}$ by $F(p_s)$. We show that $F(x) \leq F(y)$ for $0 < x \leq y < 1$. We need to show that

$$\frac{x}{1-x} \leq \frac{y}{1-y} \text{ or,} \tag{2.7}$$

$$x(1-y) \leq y(1-x) \text{ or,} \tag{2.8}$$

$$x - xy \leq y - xy \text{ or,} \tag{2.9}$$

$$x \leq y \tag{2.10}$$

which is true by assumption. ■

Choosing a local threshold: Given the global threshold, r^*, we need to derive an appropriate local threshold; each producer would estimate r with respect to each keyword pair and if this r is lower than the local threshold one can deem that those keyword pairs are not of global interest. In order to retrieve all the discriminative pairs of global interest (those that would have been detected if all data was available centrally) we need to be conservative i.e., the choice of the local threshold must account for the worst case scenario. By doing so, we can achieve the same precision and recall values with `BigEye`, compared to a centralized baseline (discussed in § 2.2). We point out here, that one may experience an outlier case, where there are no (zero) tweets with a keyword pair in window $(k-1)$ but a significant number in window k; to avoid the divide by zero possibility, we assume that each pair appears at least once at each time window; this fix has almost no influence on the ratios that we are trying to compute.

Given the above threshold and based on the following theorem, we choose the local threshold to be $\frac{r^*}{m}$ if there are m producers.

Theorem 2 *If a keyword pair has a global ratio of occurrence r which is $\geq r^*$, the* local *ratio of occurrence of that keyword pair must be larger than or equal to $\frac{r^*}{m}$ at one or more of the m producers.*

Proof. Let the number of occurrences of a keyword pair in the current and previous time windows be N_k^z and N_{k-1}^z, respectively. The ratio of occurrence of the pair at the global level $(\frac{N_k^z}{N_{k-1}^z}) \geq r^*$.

Case 1: $\frac{N_k^z}{m}$ ***is an integer***:

Let the number of occurrences of the pair in the current time window (window k) at the local producers be the following:

$\frac{N_k^z}{m} + c_1$, $\frac{N_k^z}{m} + c_2$,..., $\frac{N_k^z}{m} + c_m$, where c_i and $\frac{N_k^z}{m}$ are integers.

The summation of all the occurrences, across all producers should be equal to the global count N_k^z, i.e., $(\frac{N_k^z}{m} + c_1) + (\frac{N_k^z}{m} + c_2) + ... + (\frac{N_k^z}{m} + c_m) = N_k^z$.

Thus, $m\frac{N_k^z}{m} + \sum_i c_i = N_k^z$, hence, $\sum_i c_i = 0$

Case 1A: P_i with $c_i \geq 0$.

The ratio of occurrence of the keyword pair of interest at P_i is $\frac{\frac{N_k^z}{m}+c_i}{N_{k-1}^{z'}}$, where $1 \leq N_{k-1}^{z'} \leq N_{k-1}^z$. Here, $N_{k-1}^{z'}$ is the number of occurrences of the keyword pair in the previous time window (window $(k-1)$) at P_i; naturally this is $\leq$ the global count in that window.

The next step shows that the theorem holds regardless of the value of $N_{k-1}^{z'}$.

$\frac{\frac{N_k^z}{m}+c_i}{N_{k-1}^{z'}} = \frac{N_k^z}{mN_{k-1}^{z'}} + \frac{c_i}{N_{k-1}^{z'}} \geq \frac{N_k^z}{mN_{k-1}^{z'}} \geq \frac{N_k^z}{mN_{k-1}^z}$.

But $\frac{N_k^z}{mN_{k-1}^z} \geq \frac{r^*}{m}$ and hence, the local rate of occurence at P_i is higher than $\frac{r^*}{m}$.

Case 1B: P_i with $c_i < 0$.

Since $\sum_i c_i = 0$, there there must be at least one other producer with $c_l > 0, l \neq i$; Case 1A will now apply to that producer l.

Case 2: $\frac{N_k^z}{m}$ ***is not an integer***:

If all occurrences at local producers are $\lfloor \frac{N_k^z}{m} \rfloor$, their summation becomes smaller than N_k^z. Hence, the number of occurrences with respect to at least one of the producers, denoted as P_i, must be $\geq \lceil \frac{N_k^z}{m} \rceil$. In other words, the ratio of occurrence at P_i is $\frac{\lceil \frac{N_k^z}{m} \rceil}{N_{k-1}^{z'}}$, where $1 \leq N_{k-1}^{z'} \leq N_{k-1}^z$. Similar to the previous case, $\frac{\lceil \frac{N_k^z}{m} \rceil}{N_{k-1}^{z'}} \geq \frac{r^*}{m}$

■

Distributed event detection algorithm: Based on the above findings, `BigEye` applies the following algorithm for distributed event detection.

1. Each producer computes the ratio of occurrences of keyword pairs available locally and transmits the pairs having ratios larger than or equal to $\frac{r^*}{m}$ to the summarizer.

2. The summarizer sends a list of the received pairs to all the producers and inquires about the occurrences of those pairs at the producers. Any producer that had identified that keyword pair, but had not reported it (because it did not meet the threshold) now reports the number of occurrences of that pair. Once this information is available, the summarizer computes the global ratios of all pairs received in step (1).

3. The summarizer filters out pairs with the global ratios less than the global threshold, r^*.

Theorem 2 proved that any globally significant keyword pair "will" be reported by at least a single producer in the first step above. This proves the following lemma.

Lemma 3 `BigEye`*'s distributed detection algorithm achieves 100% precision and recall, relative to centrally available data.*

Discussion: The performance of our algorithm degrades when the local threshold is very small. When the local threshold becomes very small, the number of keyword pairs sent by the producers to the summarizer increases drastically. Specifically, this happens when the number of producers m is very large or the global threshold r^* is very small, or both. When the local threshold becomes very small (say has a value 1) each producer sends all the pairs; to avoid division by zero we had implicitly set the ratio of occurrence of any pair at a local producer to be greater than or equal to 1. However, in practice, these cases are not of interest. A very large set of producers will imply that the local data consists of small sets, and thus, it will be hard to detect events that are of global interest. A very small threshold will also fail in discriminating between key events of interest and others.

2.5 Event Consolidation

Different discriminative pairs detected by the summarizer (based on local reports from the producers) may refer to the same physical event. This is because a single event can be characterized by multiple discriminative keyword pairs. We demonstrate this phenomenon using an example. The discriminative keyword pairs (boy, drowning) and (boat, rescue) could refer to the same event where a drowning boy in a river, was rescued by a fishing boat. Because these allude to the same event, we need to merge the key word pairs,

and thus avoid unnecessary retrieval of redundant visual summaries pertaining to this same physical event. For merging similar events, `BigEye` packs microblogs containing specific "keyword pairs" into clusters. To consolidate two keyword pairs representing the same event, the similarity between two clusters represented by these keyword pairs is computed. If the similarity score is larger than a given threshold both the clusters pertaining to the keyword pairs are consolidated.

Assessing this similarity in a distributed setting is challenging. Naively sending the entire cluster of words associated with a keyword pair to the summarizer for computing similarity scores defeats the purpose of reducing communication costs. Thus, `BigEye` consists of an approach to consolidate events across producers, in bandwidth constrained distributed environments. The approach consists of two steps described below.

Step I: In the first step, `BigEye` tries to consolidate keyword pairs representing similar events at the local producers. Specifically, it consolidates events corresponding to "keyword pairs," the clusters associated with which have content that are *very similar*. It uses the Jaccard distance [185] to measure the similarity between the two clusters (events). Previous work [259] has reported that the Jaccard distance outperforms other similarity metrics for event consolidation in this way.

Returning to our earlier example, "a drowning boy was rescued by a fishing boat in the river" have the discriminative keyword pairs (boy, drowning) and (boat, rescue). One can expect that the similarity score between the two local clusters of the corresponding discriminative keyword pairs (boy, drowning) and (boat, rescue) to be high (we find such scores to be consistent with what is observed in the case when all data is available centrally).

Based on this, the similarity between clusters of microblogs is computed at each local producer. If the distance between any two clusters of events exceeds a certain threshold with regards to the Jaccard distance, the local producers notify the central summarizer that they should be consolidated.

Note here that such approaches (although not identical to what we propose) have been previously used in event detection [79, 259]. In our case, we start by having a number of clusters equal to the number of detected keyword pairs; after the consolidation phase, we end up having only C clusters of microblogs representing the physical events. Our algorithm is similar to the one used in [259] for consolidating similar events.

To assess the similarity between microblogs belonging to two clusters, we measure the similarity between the words belonging to the two sets of microblogs. Specifically, we measure the ratio of the number of unique words that are present in "both" sets of microblogs to the total number of unique words in both sets. This metric referred to as Jaccard similarity [185] has been reported to outperform other metrics in assessing the similarity of datasets, and in particular for event consolidation. [259]. At summarizer, if the majority of the producers ($\geq 50\%$) indicate that two keyword pairs should be consolidated, the summarizer sends feedback to all the producers to merge the contents associated with these keyword pairs. This helps to consolidate highly similar events at individual producers, without sending the entire cluster contents to the summarizer.

Step II: In the second step, `BigEye` further tries to consolidate global events that do not have very high similarity locally at individual producers, and were not consolidated in Step I. To achieve this objective, it seeks to only exchange minimal information with

the summarizer to limit bandwidth consumption. Specifically, it employs minHash [40] and Locality Sensitive Hashing (LSH) [101] functions at each producer, to convert a cluster of words represented by a discriminative keyword pair into a set of hash integers. minHash and Locality Sensitive Hashing (LSH) are techniques commonly used to measure the similarity of large documents within reasonable running times ([101, 144]). The probability that hashes of two sets are similar is equivalent to the corresponding Jaccard similarity of the same sets [23, 144]. Each producer transmits the computed hash values to the summarizer. The summarizer compares the hash values across clusters to measure the similarity globally. The bandwidth consumed on sharing the value generated by minHash is significantly smaller than the bandwidth consumed on sharing the entire cluster to the summarizer (as will be shown in § 2.7). At this point, `BigEye` has tried to reconcile the possibility that a single global event of interest was perhaps identified as different events because there were multiple keyword pairs from tweets that were used as discriminatory features for this event. While we are not able to completely eliminate a single event being wrongly classified as multiple events, this process drastically reduces the possibility.

2.6 Composition of visual summaries

The final module of `BigEye` deals with the retrieval of a set of multimedia objects (e.g., images) from producers to visually summarize an event; the goal is to achieve this retrieval in the minimum amount of time. Without loss of generality, we assume that the number of objects (fixed) required to compose a visual summary for each detected event, is set a priori (e.g., by a consumer) and that this number is the same for all events of

interest. Without a priori knowledge of what events occur, this ensures that every event is summarized fairly, with no priority of one event over others. However, by applying proper weights it is easy to extend the approach to account for cases wherein events are to be prioritized.

A summary should include multimedia objects that best describe the corresponding event of interest. Thus, we assign a score to each object based on a set of features (e.g.,retweet count, favorite count etc.). We denote this set of selected features as $f_1, f_2, \ldots, f_F$ and the score associated with each item is equal to $f_j = \gamma_1 f_1 + \gamma_2 f_2 + \cdots + \gamma_F f_F$, where j is the item index. The summarizer then looks for a set of multimedia objects that maximizes $\boldsymbol{f} = \sum f_j$.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} f_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} w_{ij}^{1} x_j \leq B_i, \ i = 1, \ldots, m. \\
& \sum_{j=1}^{n} w_{ej}^{2} x_j \leq C_e, \ e = 1, \ldots, E. \\
& w_{ij}^{1} = \begin{cases} w_j & j \in P_i \\ 0 & otherwise \end{cases} \\
& w_{ej}^{2} = \begin{cases} 1 & j \in C_e \\ 0 & otherwise \end{cases} \\
& x_j \in \{0, 1\}, \ j = 1, \ldots, n.
\end{aligned}
\tag{2.11}
$$

The objects to be retrieved may be of heterogeneous sizes; w_j denotes the size of an object j. Let the bandwidth between a producer and the summarizer be B_i, where i

is the producer index. We denote the number of images to be retrieved from each cluster (event) by C_e. The total number of events is denoted as E. Since there is a bandwidth constraint imposed by the network, retrieving the images with the highest quality (score) may induce large delays. Selecting what to send from each producer to maximize the objective function while respecting the bandwidth and the event coverage constraints is an NP-Hard problem [130]. This is because the problem can be mapped to a multidimensional knapsack instance as shown in eq:knapsack. The optimization problem aims to maximize the objective function by picking up the multimedia objects with the highest scores subject to a set of constraints – the bandwidth between the producer and the summarizer, and the number of images belonging to a certain event should not exceed the limit that is imposed. This is akin to filling up a multidimensional knapsack with objects subject to constraints on the knapsack sizes.

Online algorithm. Solving the above optimization problem is not trivial for two reasons. First, as mentioned above, it is known that knapsack is an NP-Hard problem [130], and thus obtaining the optimal solution would require an exponential running time. Second, in real-world scenarios, the bandwidth allocated to a producer is not known a priori. The estimated bandwidth in the idle state is different from the available bandwidth in real time, as the producers might share the same paths and thus affect the bandwidths available to each other (in addition to traffic dynamics). Thus, the optimization problem needs to be solved online. Generally, this problem is known in the literature as knapsack of unkown capacity [66].

As evident from the problem formulation, there is a trade-off between the retrieval time and the quality of the retrieved items (the total score). We design our algorithm such that it is flexible to allow the operator to favor the quality over time or vice versa. First, each producer sends the metadata (size and score) of multimedia objects to the summarizer. Next, the summarizer sorts the objects based on a rank given by the ratio, $\frac{1+Lf_j}{w_j}$. This ratio captures the relation between the score of the object and its size. L is a normalizing factor which can allow favoring one metric (e.g., score) over the other as desired by the deployer. For example, if $L = 0$, the algorithm retrieves the objects with the smallest sizes which lead to shorter retrieval times but can cause poor quality objects to be retrieved.

Initially, when no bandwidth estimates are available, `BigEye` obtains an initial estimate of the bandwidth between each producer and the summarizer, using the *iperf* utility [11]. Next, the algorithm updates this bandwidth estimate during execution as follows. It considers a fixed period of time (denoted by T), and the summarizer retrieves the maximum number of multimedia objects from each producer, that are retrievable within this period. Partially retrieved objects (complete retrieval not possible in T) are not counted. Based on the objects retrieved and their sizes, from a producer, the summarizer estimates the bandwidth to that producer (data retrieved divided by T).

At any given point, the summarizer has the sorted objects as described earlier, and the bandwidths to each of the producers. It then tries to pull up an object from producers in order from the ranked list. For each object, the summarizer first checks if the object is still relevant (meaning that the necessary number of objects have already been received for the corresponding event). If not, the object is discarded. Otherwise, it checks if the

retrieval of the object violates the bandwidth constraints to the corresponding producer (i.e., can the object be retrieved within T seconds, given the estimate of the bandwidth to that producer). If there is no violation, the object is retrieved from the producer and the bandwidth to that producer is decreased by the size of the object. In particular, if the estimate of the available bandwidth to producer P_i was B_i and the object size was w_j, $B - i$ is updated to $B_i = B_i - w_j$. Further, the number of objects that are needed for the summary corresponding to that event (say e) is decreased by 1 (i.e., $C_e = C_e - 1$).

For each new time period, the producer's bandwidths are implicitly updated during the above process. The summarizer repeats these steps until the total number of required objects are retrieved for all events ($\sum_e C_e = 0$). The procedure is formally summarized in Algorithm 1.

2.7 Implementation and evaluations

The implementation and experimentation environments are described first in this section. Subsequently, our evaluations of `BigEye` are provided. Each module is evaluated separately to showcase the benefits of each. We consider the holistic performance of our system at the end of the section. To emulate a network, we use Mininet [173], a popular software defined network (SDN) emulator. To show the realism of our approach, we use the NDN test-bed topology to represent the network [178]. This topology is used in a real deployment where distributed producers are connected. We change the default latencies and bandwidth per link in some experiments to showcase and demonstrate the scenarios of interest (bandwidth constrained environment).

We use the ONOS SDN controller [203] to manage the communication and routing between producers and the summarizer (and vice versa). Each producer receives microblogs (e.g., tweets) of interest, and by using `BigEye`, the necessary computations and communications are performed at both the producer and summarizer sides, as described in the aforementioned three stages of the system. Finally, `BigEye` outputs a summary of the global events of interest plus a concise visual summary (the holistic output of the approach is discussed in § 2.7.7)

2.7.1 Datasets collection and distribution

We collect tweets using Apollo [21], a framework that retrieves information from Twitter using Twitter API. The framework allows its users to collect tweets that match keywords of interest such as "disaster" and "wild fires". The collected datasets are cleaned by removing (1) re-tweets, (2) stop words and special characters [206] and (3) URLs. Subsequently, we apply stemming [201] to the collected data. We note that this is a common practice in data mining applications. We use Python-NLTK tokenizer [241] and the Porter stemmer [201], which are common tools used for these purposes. The collected datasets are summarized in the following:

- **Protest.** This dataset is collected using the keyword "protest". The dataset has tweets related to protests and it was collected from *March 18, 2018* to *April 18, 2018*. It consists of approximately 300K tweets after applying the aforementioned filtering methods.

- **Florence.** The dataset contains information about the Hurricane that occurred in the Carolinas in 2018. The dataset was collected using the keywords "hurricane" and "florence" from *Sept 14, 2018* to *Sept 24, 2018*. It contains approximately 100K tweets after conducting the aforementioned filtering methods.

To emulate a scenario in real time, the data is streamed and fed to the producers with respect to the time stamps collected from the tweets. For the purpose of experimentation in this paper, we choose a window size of 24 hours. The term *instance* refer to the data corresponding to each such window size. Thus, for the above datasets, we have 40 instances in total. The datasets are distributed over multiple producers with two different scenarios that are described below.

1) Natural distribution: In practice, the tweets are posted by Twitter users from different geographical locations. One can use those locations to cluster the tweets into different geographical zones, which can be later associated with the producers. However, not all tweets contain associated geolocation information; in fact less than 2% of tweets have this information [49]. This makes the problem of simulating exact real-word geographical distribution of tweets hard. So, the addresses of users (which can be obtained from users profiles) are used to mimic the geographical distribution of tweets (tweets from the same user are assumed to be made from his/her profile's geolocation information). Using this approach, we are able to retrieve 70% and 60% of this information, from users associated with the Florence and Protest datasets, respectively. We are unable to decipher the rest because some users had provided vague addresses (e.g., "space" and "floor"). We use an API provided by *HERE.com* to convert the provided addresses to geolocations [104].

As mentioned earlier, we used the NDN topology which reflects a real world network (real server locations); we assume each tweet is sent to the nearest physical producer (assuming that each server is a producer). Fig. 2.3 shows an example demonstrating the approach. We extend the same idea when a variable number of producers (different from the original number of servers in the NDN-topology) is considered,from as follows. First, each tweet is assigned to its nearest producer and among all producers we select the top m (ones that receive the most number of tweets) producers. Next, we assign tweets to nearest producer from these top selected m producers. For tweets where we could not retrieve the geographical locations, they are assigned randomly to a producer.

2) Synthetic distribution To further evaluate the performance of `BigEye`, we consider uniform and skewed distribution of tweets across multiple producers. **(i)** Skewed distribution: The distribution of the data over the producers follow a Gaussian distribution with $\mu = \frac{N_k}{m}$ and $\sigma^2 = \beta * \mu$; we vary the β to control the skew. **(ii)** Uniform distribution: The distribution of the data over the producers follows a uniform distribution.

We also collect the images associated with the tweets and stream them on the producers according to the selected distribution. For tweets scores, we select three features (f_j as discussed in § 2.6): retweet count, favorite count and follow ratio defined as $\frac{\#\text{ of followers}}{\#\text{ of following}}$[2]. We also select L to be 1. Twitter allows users to post more than one multimedia object in a single tweet. We treat each as independent object while giving each the same score as their associated tweet. Twitter API keeps multiple versions of the videos each with different quality. To avoid redundancy, we pick only a particular version randomly.

[2]Assessing how good the selected features are in practice is beyond the scope of this paper.

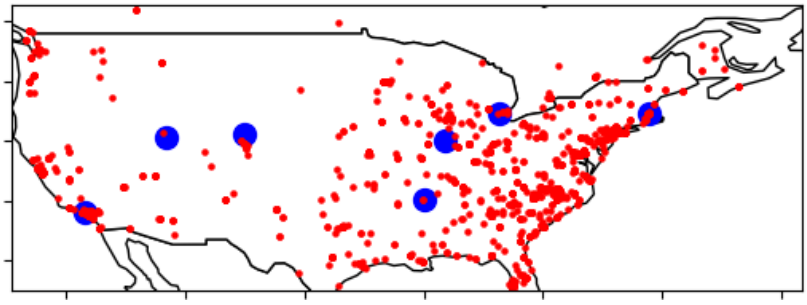

Figure 2.3: A map of the United States, representing two hours of tweets collected from the Florence dataset. Blue dots represent producers and red dots represent the locations of tweets. Each tweet (red dot) is sent the nearest producer (blue dot).

2.7.2 Evaluation parameters

Our evaluations of the performance of `BigEye` consist of a comparison with an approach wherein all data is made available centrally. Our evaluations are on the datasets that were discussed in the prior section. We stream each dataset separately, in line with our assumption (streaming data is aligned with the scope of a realistic scenario) in § 2.3. The results that we obtain are then integrated before we discuss them for two reasons. First, we find that the performance with `BigEye` is consistent across both datasets, when the data is distributed over geographically separated producers, compared to when the data is centrally available. Second, the generality of `BigEye` ensures that its behavior with any crowd-sensed dataset, belonging to a particular scope, is consistent. As a consequence of these two reasons, an independent discussion of the details of the performance of each dataset provides no new information, and is wasteful of space.

Our experiments are performed with all instances from all of the datasets (40 in total), considered one at a time. For statistical significance, experiments are repeated 50 times. `BigEye`'s three components are integrated holistically; however, first we indepen-

dently evaluate each component to provide microscopic views of the benefits of each. The following describe for these module specific set ups.

Distributed Event Detection Module: As described in § 2.4, we find that our method always yields the same precision and recall with respect to detection of key events, as that of an ideal system which considers that all data can be made available centrally. We experimentally validated this and do not further showcase the event detection accuracy in the interest of space.

To evaluate if the number of keyword pairs that are returned by `BigEye` to the summarizer is reasonable, we compare the number with the best case scenario. In particular, we consider an *oracle* that does a brute force search, considering all possible subsets of the keywords pairs sent (the ranked orders of those pairs are still maintained), and checks if any of those subsets yields the same precision and recall as our approach (and the central approach). We choose the smallest subset among these as the best possible scenario (we label it as oracle prediction in the results that we present). In other words, instead of choosing a threshold $\frac{r^*}{m}$, we find a the smallest value $l \leq m$ such that choosing $\frac{r*}{l}$ results in the detection of all events of interest.

Consolidation of events: Our proposed distributed consolidation is evaluated next. Specifically, `BigEye`'s approach is compared with a case all the data associated with the clusters is sent to the summarizer which then applies a consolidation. We denote the baseline as *central consolidation*. We point out that this is the consolidation used in the prior work Storyline [259], where the data is centrally made available and similarity assessment is based on Jaccard distance.

The metrics of interest are accuracy (defined next) and the amount of data sent from the producers to summarizer for the purposes of consolidation. Accuracy is defined to be the ratio of the number of keyword pairs that are grouped correctly (the events are correctly consolidated) to the total number of keyword pairs. Two keyword pairs are incorrectly grouped if (a) these pairs belong to the same event but are put in different groups and (b) if they belong to different groups (events) but are consolidated into the same group. We also compare the amount of sent data from the producers to summarizer with our approach (in bytes), with the other approach.

Composition of visual summaries: We assess the "quality" from `BigEye`'s visual summary module with respect to the centralized case (where no bandwidth constraints are present). In particular, if data is located centrally, the summary can be composed using the objects that have the highest associated scores. However, objects with the highest scores are not always retrieved in `BigEye` to meet timeliness constraints as discussed in § 2.6. To evaluate the impact of this, we evaluate the quality which is defined as the the ratio of the summation of the scores of the multimedia objects retrieved with `BigEye`, to the corresponding sum score achieved in the centralized case.

We also evaluate the delay in retrieving the visual objects. Formally, this delay is defined as the time it takes for the summarizer to receive the data used for the visual summary from the producers. Specifically, the metric we use is the average delay incurred in receiving 1 MB of data (we normalize this since the sizes of the visual summaries could be different for different events). We compare the delay of `BigEye`'s visual summary module with the following baseline (denoted as *baseline*). Each producer is assigned the job of

sending information related to a specific event. If the number of events are more than the number of producers, the events are evenly distributed across the producers. If a producer is assigned more than an event, the resources are shared equally between them with no priority given to one over the other. Our goal here is to showcase the efficacy of our parallelization approach with a second approach (the baseline) that also parallelizes transfers but is not bandwidth-aware (does not take into account the different bandwidths to the different producers).

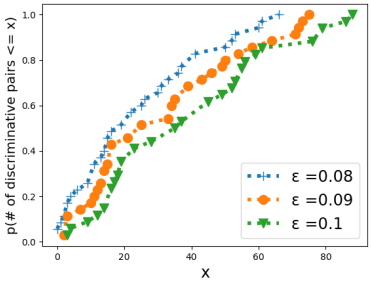

Figure 2.4: CDF of the number of detected discriminative pairs with different global thresholds.

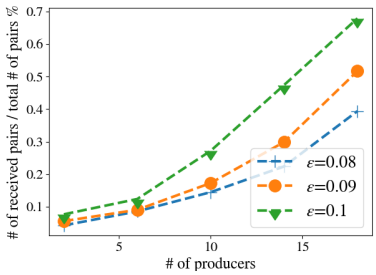

Figure 2.5: The performance of our distributed event detection with varying number of producers

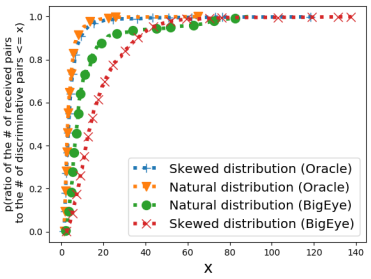

Figure 2.6: Performance of our distributed event detection with different data distributions

2.7.3 Results on distributed event detection

First, we evaluate our distributed event detection module using the aforementioned metrics. We recall our discussion in § 2.4 ($H(Y|s_z)$ was to be a small value ϵ), and select ϵ to be 0.08, 0.09 and 0.1 (r^* = 100,87 and 76, respectively). Here, we also refer the reader to Table 3.1 since the notation therein is used in the discussion.

Effect of r^* on the total number of pairs retrieved by the summarizer. We plot the CDFs of the number of events (corresponding to identified discriminative key

word pairs) detected with each value of r^* in Fig. 2.4. As one might expect, as r^* increases, the number of pairs retrieved decreases (with higher r^* only the most significant events are detected). This effect is also seen in Fig. 2.5.

Effect of increasing number of producers. In Figure 2.5, we plot the ratio of the number of retrieved keyword pairs to the total number of keyword pairs identified, versus the number of producers. We assume that the data are distributed as per the *natural* distribution. As one might expect, the number of key-words pairs returned to the summarizer increases when as the number of producers, m, increases. This is because because $\frac{r^*}{m}$ decreases i.e., a lower or more conservative (local) threshold is used at each of the producers. It is worth noting that with small $r^* = 76, (H(Y|s_z) = 0.1)$ and large $m = 18$, the total number of received pairs is less than 1% of the total number of keyword pairs considered globally.

Comparison with oracle. Next we examine how the number of keyword pairs retrieved with `BigEye` compares to what is obtained by an oracle, when the data is spread across the producers as per the different data distributions (discussed in subsec:datasets). For the skewed distribution, we choose $\mu = \frac{N_k}{m}$ and $\sigma^2 = 0.5 * \mu$; this ensures a high skew. We fix $\epsilon = 0.09$ (r^*=87), and m to be 10.

In Figure 2.6, we plot the CDF of the ratio of the number of pairs received at the summarizer to the number of global discriminative pairs with both `BigEye` and the oracle based approach described earlier. We see that the performance of `BigEye` is similar to that of the *oracle* when data is dispersed as per the natural distribution. However, when the distribution is skewed, the performance of the `BigEye` degrades compared to the oracle.

This is because the producers with large numbers of tweets have a large number of keyword pairs that pass the conservative threshold selected by `BigEye`; thus, they end up sending a large number of pairs that are not useful in detecting key events.

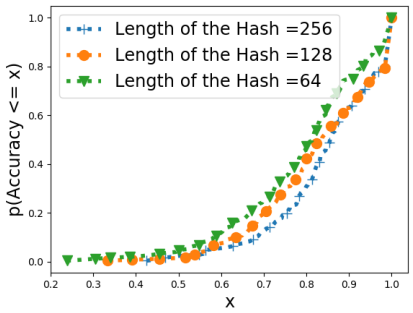

Figure 2.7: Distributed consolidation accuray with respect to centralized consolidation (Storyline [259])

.

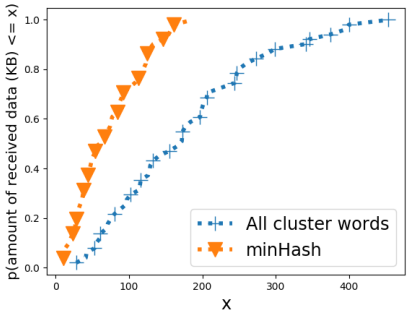

Figure 2.8: Bandwidth savings from `BigEye`'s distributed consolidation approach in terms of total amount of data sent from producers to summarizer.

2.7.4 Results with regards to consolidation

Next, we evaluate the benefits from `BigEye`'s consolidation module. In our evaluations we use the same ϵ values mentioned earlier in § 2.7.3. Recall that `BigEye` consolidates events over two steps. In the first step, we consider a similarity requirement of 0.99 (Jacquard distance) to consolidate events at individual producers. For the second step, we consider three minHash signatures of length 64, 128, and 256 integers. We consider different minHash signatures as it has been reported that the length of generated minHash signatures affects the similarity scores [144]. We also vary the consolidation thresholds that are used centrally from 0.4 to 0.9 with a stepsize of 0.1.

In Figure 2.7, we compare `BigEye`'s consolidation approach with different minHash lengths. We observe that the similarity estimation improves as the length of the minHash signature increases (less collisions). Similar phenomena have also been reported in previous literature [144]. We observe that, for a minHash signature of length 128, almost 85% of instances show a consolidation accuracy of larger than 70% of what is achieved if all data was available centrally. `BigEye` provides consistent consolidation accuracies irrespective of the number of producers.

We show next the consolidation benefits in terms of the reduction in the volume of data sent from producers to summarizer in bytes, compared to sending all keywords in the clusters (needed for central consolidation); Fig. 2.8 shows that our approach reduces the communication costs significantly compared to that baseline approach, and in particular the average cost by 60% (fewer bytes).

2.7.5 Results on visual content retrieval

Next, we evaluate the performance of `BigEye`'s visual summary module with respect to the baseline approach discussed in § 2.7.2.

Delay performance with different numbers of producers. We evaluate the performance of `BigEye`'s final module with different numbers of producers (we consider 2, 6 and 10 producers). We assume that the data is distributed uniformly over these producers. We plot the CDFs of the average delay (over the instances) with `BigEye` and the baseline in Figure 2.9. First, we observe that our approach outperforms the baseline by approximately 67% on average. Furthermore, we also see that the delay decreases as the number of producers increases initially. This is because of the increased parallelization

in retrieval that is possible due to this. However, the benefits reach a point of diminishing gains. Specifically when the number of producers increases beyond a certain number (e.g., above 6 and 10 in Figure 2.9) because of limitations in the NDN network structure very little additional parallelization is possible (the producers share common paths). We have manually constructed a tree network where each producer is connected to the summarizer with a dedicated link. In this case, we do observe significant performance enhancements between the cases of 6 and 10 producers since parallelization is now viable. We omit showing these results to conserve space.

Impact on summary quality. We plot the CDF of the quality of the visual summaries obtained with different numbers of producers versus the baseline approach in Fig. 2.10. We see that the best quality is achieved when there are only two producers. Typically, if all data is at a single producer, it is natural to pick the objects with highest scores ($\frac{f_j}{w_j}$) according to the global sorted list shown in Algorithm 1. However, when the number of producers increases and we try to achieve parallelization of transfers, the highest ranked objects in the global list are not always retrieved. In particular, if a single producer has most of the highest ranked objects, we will retrieve objects of inferior quality from other producers. In other words, the summarizer pulls other objects (with lower rank) from other producers, which in turn affects the achieved quality (the trade-off is the reduction in terms of the delay).

Data requirement to compose summaries. Next we evaluate the performance of `BigEye` when the user imposes different requirements on the number of objects that are needed in the visual summary. Specifically, we impose requirements of (25%, 50%, 75%) of

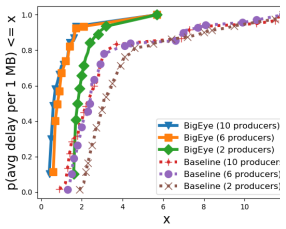

Figure 2.9: Delay in visual content retrieval with different number of producers.

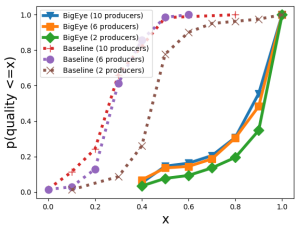

Figure 2.10: Summarization quality of visual summary with different number of producers.

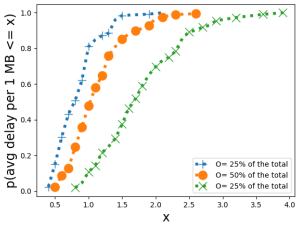

Figure 2.11: Delay in visual content retrieval with different number of objects required to compose summary.

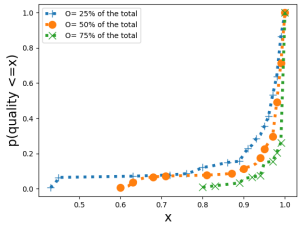

Figure 2.12: Summarization quality of visual summary with different numbers of required objects

the total number of multimedia objects (corresponding to each detected event) to compose the summary. We use the skewed data distribution with $\beta = 1$ (very skewed distribution). This means that a small subset of the producers has a large number of multimedia objects, while others have only few. We fix the number of producers to be 6. As shown in Fig. 2.11, when the required number of objects is only 25% of the total, the lowest delay is achieved. As evident, as this requirement increases, the delay increases. In fact, because of the skewed distribution, when we need 75 % of the objects, the summarizer is forced to abandon parallelization and retrieve all objects from the producers with the large number of objects. This drastically affects delay.

Fig. 2.12 shows, as one might expect, that the quality of the composed summary improves as the number of required objects to compose the summary increases. However this improvement is not drastic. This is because the first objects that are retrieved have the highest scores; later objects contribute less and less to the quality of the summary.

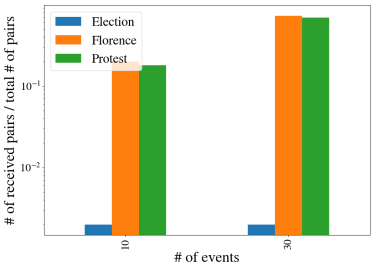

Figure 2.13: Performance of the dirstibuted event detection module with multiple datasets (scalable setting).

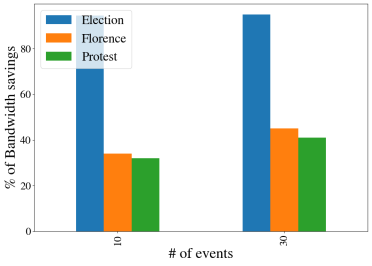

Figure 2.14: Bandwidth savings from `BigEye`'s distributed consolisdation with multiple datasets (scalable setting).

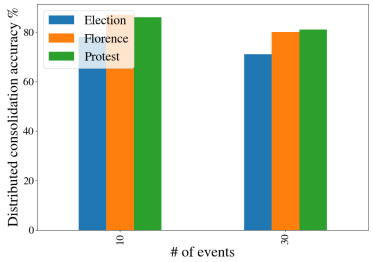

Figure 2.15: `BigEye`'s distributed consolidation accuracy with multiple datasets (scalable setting).

2.7.6 Scalability

To showcase `BigEye`'s scalability where the generated data is much larger, we use a dataset collected on election day in the United States. It consists of around six million tweets and we clean it up as we discussed earlier in § 2.7.1. We point out here that the datasets used in the results shown earlier (Florence and Protest) are small in comparison to the Election dataset (by a factor of about ≈ 20). We select global thresholds such that an approximate constant number of top events (10 and 30) are retrieved, and assess the performance of our modules. Specifically, we examine how the results scale with the dataset size (in comparison to the smaller ones). We point out here that tuning the global threshold to get the exact same number of events across all datasets is hard; thus we choose thresholds such that the ratio of events retrieved across each pair of datasets varies between 0.9 to 1.1 (i.e., the numbers are approximately 10 or 30 but not exact). We select the number of producers, m, to be 10 and a hash of length 128. The accuracy of our distributed event detection module remains at 100% like in the case where all the data is available centrally (similar to the results reported earlier) even when the dataset is large. The ratio

of the recieved keyword pairs (after processing by `BigEye`) to the total number of keyword pairs is drastically smaller with the Election dataset (only 0.002%) in comparison with Florence hurricane and Protest datasets as shown in Figure. 2.13. One might expect this to be the case since the total number of keyword pairs is much larger in comparison to the candidate discrminative keyword pairs (with a local ratio of occurrence larger than $\frac{r^*}{m}$, the local threshold); recall that only these candidates are sent from the producers to the central controller. In large datasets (e.g., the Election dataset), the total number of keyword pairs is much larger than the correponding number in smaller datasets (e.g., Florence); since we choose a target number of top events to be detected, the ratio of candidate discriminative key word pairs to the total number of keyword pairs becomes drastically smaller. Thus, the takeaway is that one can expect even further reductions in overhead with `BigEye` as the dataset sizes grow, because of large reductions (if we seek to detect a certain target number of top events).

We next evalute our second module (consolidation) with a large dataset in terms of both bandwidth savings and consolidation accuracy. With the new larger dataset, the number of words that mapped on to each event increases significantly; thus, `BigEye`'s approach of sending a fixed hash representation of the event contents (as compared to sending the raw words corresponding to the event) reduces the overall proportion of transmitted data in comparison to the smaller data sets as shown in Figure. 2.14. However, as shown in Figure. 2.15, we see that with the Election dataset, we have a slighlty lower consolidation accuracy in comparison with the results obtained with the other datasets. This is because the used hash length is small (only 128) and hence it is not sufficient to capture the sim-

ilarity between the detected events in this large dataset. As indicated earlier, a possible way to enhance the consolidation accuracy is to increase the hash length (see Figure. 2.7). This is because the similarity estimation improves as the length of the hash increases [144]. To verify that this is applicable on large datasets, we increase the hash length to 256 and run `BigEye` with Election dataset and we achieve consolidation accuracy of 90% when the number of events equal to 30.

2.7.7 Holistic output of BigEye

Next, we show the holistic output of `BigEye`. We select random instances from three datasets. We use the previously mentioned Protest and Florence datasets, and a new dataset (called disaster dataset [127]) that is available for public use to further show the merits of our proposed framework. The summaries are composed of (1) the discriminative key word pairs, (2) textual summaries (complete tweets) describing the observed events and (3) the visual summaries (we select the best images to avoid cluttering). The discriminative pairs and the textual summaries of the three datasets are shown in Tables 2.2, 2.3 and 2.4, and the corresponding visual summaries are shown in Figures 2.16 to 2.18. To elaborate, we'll go through a sample of the results associated with Protest dataset. On March 30, two major protests were detected. The first related to teachers in Kentucky schools, who were protesting against a pension bill which forced schools to close (an image of the protest is shown). The second event is related to a protest in Gaza where seven people were killed and dozens were injured. One can easily understand and differentiate between these by

following the textual summaries of the events and their corresponding visual summaries.

Event I on Sept 15

Event I (left) and Event II (right) on Sept 16

.

Figure 2.16: Florence dataset

Event I (left) and Event II (right) on March 29

Event I (left) and Event II (right) on March 30

Event I on March 31

Event II on March 31

Event III on March 31

Figure 2.17: Protest dataset

2.8 Related work

Postings on social networks have been recently used as sensor outputs [256, 257] that can be used to discern events. There exist some prior studies on detecting events

Event I on Jan 14

Event II on Jan 14

.

Figure 2.18: Disaster dataset

from such sensors' data (with goals aligned with those of `BigEye`). Allan et al. [17], used "term frequency" (tf) and "inverse document frequency" (idf) features to build a query representation for content from news stories and identified an event, when the similarity score of new news story was less than a given threshold in comparison to any previous news query in memory. Similarly, Shamma et al. [217], used a normalized term frequency to identify peaky topics, the terms which are particular to a time window, to detect highly localized events of interest. Benhardus et al. [30], also used tf-idf analysis and relative normalized term frequency analysis, on twitter documents to identify trending topics. However, these approaches were reported to be inefficient in differentiating between separate event instances [259]. Moreover, unlike tf-idf, `BigEye` works by only computing information gain over two consecutive time windows.

Text stream clustering has also been applied for event detection. Ordonez et al. [192], Zhong et al. [279] and Aggarwal and Yu [8], used optimizations of k-means algorithms to cluster data streams for events detection. Similarly, communication patterns [50], social network topological features [7], language specific features [87, 239, 261], and location of tweets [39, 145, 253] have also been used by researchers for clustering data to detect events.

Nevertheless, precisely defining the number of clusters (k) for online streaming data is not feasible. Researchers have also used topic modeling for event detection [110, 142, 281]. However, topic based approaches have been reported to be inefficient in identifying events happening in parallel instances [259]. Unlike these methods, `BigEye` detects events by measuring the temporal bursts in the word-pairs that do not co-occur frequently. `BigEye`'s event detection approach is closely related to Storyline that was proposed by Wang et al. [259]. However, unlike `BigEye`, Wang et al. only focuses on event detection when data is centrally located.

A different line of work considers the problem of truth finding in social sensing blogs [148, 218]; however, these only work when all the data is made available centrally unlike `BigEye`. In contrast, `BigEye` is targeted for a distributed setting, wherein the data is distributed across multiple producers that are geographically separate.

`BigEye` centers around the detection of global events by only sharing minimal amounts of information between distributed producers and a central summarizer. There have been some prior works on selectively sending information to a central entity [98, 132, 162]. There are also related works focusing on calibrating distributed sensors with the objective of finding global measurements from those sensors (e.g: measuring urban air pollution) [175,268]. However, unlike `BigEye`, these approaches do not focus on event detection. Closely relevant to our study is the study by McCreadie et al. [167]. Unlike `BigEye`, they do not consider bandwidth constraints and only try to minimize the event detection time by distributing the computational costs of processing documents across multiple machines.

2.9 Conclusions

In this paper, we address the important problem of detecting global events from crowd-sensed data. Towards this, we design and implement `BigEye`, a system that enables (a) the detection of key global events based on distributed crowd-sensed data that exists at geographically spread out producers and (b) the crafting of visual summaries that provide concise zoomed-in views of such events. `BigEye` distinguishes itself in that it is extremely thrifty in terms of the bandwidth that it consumes, i.e., very little of the raw crowd-sensed data from the producers needs to be transferred to a central entity for both event detection and the subsequent visual summarization. In spite of its thriftiness, it is able to achieve 100 % precision and recall compared to approaches where all crowd-sensed data is made available centrally. Via emulations of realistic scenarios, we show that `BigEye` only consumes 1 % of the crowd-sensed data for detecting key global events, and its parallelization of visual content retrieval reduces the average delay by 67 % compared to baseline approaches.

Table 2.1: Key notation

Symbol	**Description**
N_k	# of tweets in time window k
N_k^z	# of tweets in time window k that contain the pair s_z
p_s	$\frac{N_k^z}{N_k^z + N_{k-1}^z}$
k	index of the data stream window
$\in$	$H(Y) - threshold$
m	# of producers
i	Producer index
P_i	i^{th} producer in the the system
r	ratio of occurrence of keyword pair s_z in time window k to the corresponding occurrence in time window $k-1$
n	total # of multimedia objects
j	multimedia object index
w_j	size of the j^{th} multimedia object
f_j	score of the j^{th} multimedia object
B_i	Bandwidth between P_i and the summarizer
C_e	# of multimedia objects inquired from event e

Algorithm 1 Multimedia objects retrieval

CalibrateFunction *Calibrate*end (ReceivedObjects[i])

Bandwidth[i] $\leftarrow \sum w_j, j \in ReceivedObjects[i]$

BestBw[i] $\leftarrow$ Bandwidth[i]

for i in $\boldsymbol{I}$ **do** MaxBW[i] $\leftarrow iperf$ (P_i, summarizer) Bandwidth $\leftarrow$ MaxBw

BestBw $\leftarrow$ Bandwidth

objects $\leftarrow$ sorted (objects, $\frac{1+Lf_j}{w_j}$)

while $\sum_e C_e \neq 0$ **do**

for all j in objects **do** $i \leftarrow j \in P_i$

if Bandwidth[i]- $w_j \geq 0$ & C_e- 1 ≥ 0 **then** Bandwidth[i]= Bandwidth[i]- w_j

$C_e = C_e$ - 1

RequestObject(j, P_i)

wait (timeWindow)

Bandwidth $\leftarrow$ BestBw

Table 2.2: Florence Hurricane Summarization

Date	**Detected keywords**	**Textual summary**
Sept 15	**Event I:** sympathy, hurricane, deepest, hurricane	"My deepest thoughts and prayers are with those in North and South Carolina, and Virginia affected by Hurricane Florence"
Sept 16	**Event I:** 'abandon', 'hero', 'six', 'dog'	"Six dogs have been rescued from rising flood waters, after they were locked in a cage and abandoned by their owners"
	Event II: 'dog', '64', 'rescue', 'hurricane'	"As Florence loomed, a pet lover escaped South Carolina with 64 dogs and cats on a school bus"

Table 2.3: Protest Summarization

Date	**Detected keywords**	**Textual summary**
Mar 29	**Event I:** 'viral', 'plung', 'california', 'cliff'	"Washington state family famed for protest photo died when SUV goes off California cliff"
	Event II: 'NBA', 'office'	"Police, DSS Invade Ikeja NBA Office To Foil Protest Coinciding With Buhari's Visit"
Mar 30	**Event I:** 'bill', 'school', 'teacher', 'kentucki'	"Kentucky schools close as teachers protest GOP-passed pension overhaul"
	Event II: 'gaza', 'border', 'palestinian', 'kill', 'israel'	"7 Palestinians killed, dozens injured as Israel suppresses massive protest in Gaza"
Mar 31	**Event 1:** 'Stephon' , 'clark', 'car', 'hit'	"Sheriff's Car Knocks Down Activist During Stephon Clark Shooting Protest"
	Event II: 'student' 'Howard', 'protest'	"Howard University students stage sit-in to protest financial aid scandal"
	Event III: 'deadly', 'day', '15', 'palestinian', 'kill', 'israel'	"15 Palestinians reported killed by Israeli fire as Gaza border protest builds"

Table 2.4: Disaster Summarization

Date	**Detected keywords**	**Textual summary**
Jan 14	**Event I:**'Australia', 'forest' 'fire'	"13 years ago we predicted that the worst fire seasons would be directly observable in australia by the year 2020"
	Event II: 'volcano', 'Philippine', 'Taal', 'erupt'	"Taal volcano, located 70 km from Manila, Philippines. In addition to the eruption itself, the so-called volcanic tsunami"

Chapter 3

AcTrak: Controlling a Steerable Surveillance Camera using Reinforcement Learning

Steerable cameras that can be controlled via a network, to retrieve telemetries of interest have become popular. In this paper, we develop a framework called `AcTrak`, to automate a camera's motion to appropriately switch between (a) zoom ins on existing targets in a scene to track their activities, and, (b) zoom out to search for new targets arriving to the area of interest. Specifically, we seek to achieve a good trade-off between the two tasks, i.e., we want to ensure that new targets are observed by the camera before they leave the scene, while also zooming in on existing targets frequently enough to monitor their activities. There exist prior control algorithms for steering cameras to optimize certain objectives; however, to the best of our knowledge, none have considered this problem,

and do not perform well when target activity tracking is required. `AcTrak` automatically controls the camera's PTZ configurations using reinforcement learning (RL), to select the best camera position given the current state. Via simulations using real datasets, we show that `AcTrak` detects newly arriving targets *30%* faster than a non-adaptive baseline and rarely misses targets, unlike the baseline which can miss up to 5% of the targets. We also implement `AcTrak` to control a real camera and demonstrate that in comparison with the baseline, it acquires about *2*× more high resolution images of targets.

3.1 Introduction

Networked surveillance cameras are becoming ubiquitous. It is estimated that around one billion surveillance cameras are currently installed globally [96, 198, 252]. A standard surveillance camera has a fixed and limited view of the scene of interest depending on its orientation and the width of its lens. In many applications however, there is a need for a complete coverage of the scenes of interest (e.g., trespass prevention) and leaving parts of the scene uncovered can lead to security breaches. In addition, in many applications, high resolution/ fine grained images of suspicious targets or objects are crucially needed. Pan-Tilt-Zoom (PTZ) cameras have been considered a suitable solution satisfying these demands [26,226]. Such cameras have the ability to maneuver around, covering the entire scene unlike standard surveillance cameras. The "zoom" feature offered by such cameras, allows the camera to zoom in to acquire high resolution images of the targets and objects of interest, and yet provide wide view coverage when needed (via a zoom out). Most PTZ cameras can be controlled remotely over a network to satisfy various surveillance requirements, which

Figure 3.1: Zoomed out (left) and zoomed-in (right) views. The zoomed out view enables the detection of new targets as they enter the scene. However, it is insufficient for inferring target activities. The zoomed in view facilitates inferring target activities (i.e., a target reads a book) but only partially covers the scene. `AcTrak` balances zooming in and out such that target activities are captured, while ensuring that arriving targets are detected quickly when they step in to the scene.

opens up opportunities for building reliable and autonomous surveillance systems [18, 26]. One fundamental, unexplored question that arises in this case is: in an evolving scene, how to automatically control a PTZ camera to facilitate applications where there will be a need to (a) zoom out to identify any new target quickly when it enters the scene of interest and (b) frequently zoom in on existing targets to *be able to monitor their fine-grained activities* via high resolution images. For example, we envision scenarios with a need to capture prohibited activities (e.g., eating/drinking or photographing in museums, and smoking in shopping centers). Another application relates to deployments in shopping centers, to track target shopping interests i.e., what products customers are looking at, in addition to catching shoplifting activities. Such information would aid data analytics to understand customer behavior and/or even guide re-organizing product placements across aisles (e.g., distribute popular products to enable Covid-19 social distancing needs).

Challenges: The control algorithm require a good balance between the actions of zooming in and out. A simple example is shown in Figure 3.1. If we focus on target zoom-ins, we may miss out on arriving targets in the lower left corner and a continuous zoom out does not shed light on target activities. Achieving this balance by changing the camera's views is challenging due to two delay components associated with camera motion: mechanical camera movements incurred due to motion from one position (e.g., zoom out) to another (e.g., zoom in) and computational and networking latencies of processing and retrieving the captured frames. Therefore, a careful orchestration is needed for managing the frequency and patterns of zoom ins among targets especially in the case when several targets exist in the scene, while ensuring that we do not miss new targets.

Related Work: Multi-objective surveillance problems have been studied widely in static and steerable multi camera systems [199]. Cameras usually co-ordinate to achieve one or more objectives such as scene coverage, opportunistic acquisition of zoom-in images of targets [65], tracking [47, 204, 230, 271] or power efficient surveillance [133]. However, in our work we consider a single camera that balances the trade-off between two conflicting objectives (Figure 3.1), and previous work on multi-camera solutions cannot satisfy these objectives in a single camera setup. The closest work to ours is [65], where the authors consider a multi camera system that (1) fully covers the physical scene all times and, (2) opportunistically captures high resolution shots of existing targets. This is different from ours in two aspects. First, the physical scene coverage constraints cannot be used in our setup (when single camera is used), because when the camera zooms-in on a target, only a partial view of the scene is obtained (violating the physical scene coverage constraints).

If for cost constraints (multiple camera deployments are more expensive) the full physical scene cannot be covered, one has to manage the zoom-in shots of targets without sacrificing coverage of new targets arriving to the scene; we consider this specific case in our work. Second, we consider continuous zoom-in tracking of existing targets for activity recognition, unlike the work in [65], where the focus is on opportunistic acquisition of high resolution shots. More discussion is provided later (§ 3.5) on why a single camera solution can be desirable, and we tackle this important problem in this work. While there is prior work on controlling the PTZ of steerable cameras (e.g., [120, 182]) in a single camera setup, they have different objectives from ours; importantly, none consider changes in the target population in the scene (new arrivals) and most do not consider arbitrary motions of targets. For example, the authors in [120] propose an approach to orient the camera to zoom in on various (but fixed and pre-determined) chosen locations. Their approach allows the tuning of how frequently these locations are to be visited, but leaves other locations uncovered. Thus, this method is ineffective when tracking targets activities outside the selected locations is required. We argue that the camera should focus on activities rather than location coverage (since coverage of unoccupied locations yields little or no information).

Contributions: In this paper, we design and implement, `AcTrak`, to control a PTZ camera such that, it (a) quickly identifies dynamically arriving targets to a scene and, (b) with appropriate frequency (target specific), obtains high resolution images of each target in the scene to capture their activities at fine time scales[1]. Importantly, `AcTrak` accounts for the associated latencies when determining the camera movement.

[1]Note that we do not store biometric features of targets (for privacy reasons), but rather use high resolution shots for activity recognition.

We formulate the problem as Markov decision process (MDP), where a reinforcement learning (RL) agent that dynamically learns, and thereby determines the best PTZ configuration given the current situation. This works in an online fashion i.e., it continuously tunes the camera PTZ dynamics to cope with the evolving scene. Importantly, it tries to minimize the latencies incurred in changing the camera views, by choosing visitation patterns in an informed way. Furthermore, it does not require expensive computations to derive its next PTZ configuration during runtime (a single feed forward neural network operation is needed).

In brief, our key contributions in this paper are as follows:

1. We design `AcTrak`, a framework for the PTZ control of a steerable camera. `AcTrak` is based on MDP, which selects the PTZ configuration that provides the highest utility (discussed later) in terms of a trade-off between the goals of balancing rapid acquisition of new targets and obtaining fine grained information about existing targets, while minimizing latency penalties due to camera motion.

2. Typically, RL agents need heavyweight training to deliver high accuracy [106]. While creation of the requisite, huge number of training instances is possible on fast machines, it is time-prohibitive in our scenario since camera mechanical movements and networking and computational latencies can be in order of seconds. Hence, instead, we develop a simulator to mimic the camera, target movements and other dynamics to enable training and we deploy the trained agent during test time.

3. We showcase the merits of `AcTrak` via both simulations and real world experiments with an implementation on an off-the-shelf PTZ camera. In our real world experiments

`AcTrak` outperforms a non-adaptive baseline by acquiring *2* $\times$ more high resolution images of targets. Our simulations using public datasets show that our agent detects new targets that arrive to the scene *30%* faster than a state of the art baseline.

3.2 Our RL based Control Framework

In this section, we first describe the trade-offs between the functionalities we seek to achieve (zoom outs to find new targets and zoom in to track current targets). Next, we provide an overview of `AcTrak`, and then describe it in detail.

3.2.1 Functionalities and associated trade-offs

Scene coverage. The camera can zoom out to cover the entire scene (to find new targets) and we call this the *coverage tour*. Invoking unnecessary coverage tours at high frequencies will reduce the time for (frequency of) acquiring high resolution images of existing targets. Instead, if the camera undervalues coverage tours, new targets may leave unseen.

Zoom-ins. Upon identifying a target, `AcTrak` marks its location so that the camera can zoom in on it again in the near future, to track its activity; this obviates the need to zoom out each time in order to find that object. To acquire the high resolution image, an appropriate PTZ configuration is chosen to direct the camera at the last known (updated) location of the target. Since targets can be located at different distances from the camera, each target requires a different zoom level denoted as Z_j, where j is the target's index. Target locations are updated during coverage tours and the zoomed in visitations.

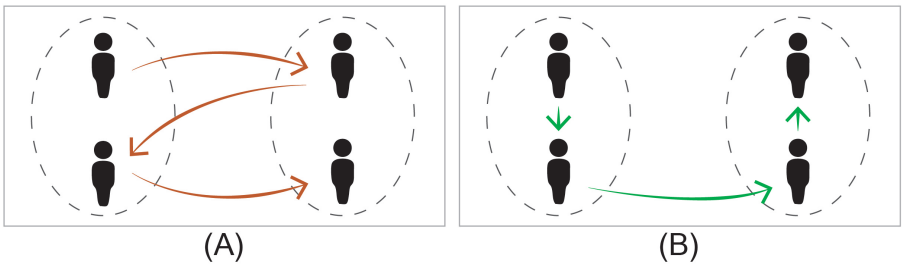

Figure 3.2: ZoomIn Scenarios. In Figures A and B, we show different camera strategies of zooming-in on existing targets (Note the arrows). The overall time to visit all targets of the strategy in Figure A in higher in comparison with the scenario in Figure B due camera's longer (wasteful) moves.

A location becomes outdated when the target moves outside its previous zoom-in view, which depends on the target's motion (e.g., walking, jogging). The camera must obtain an updated location before zooming in on the target.

Managing zoom-ins of multiple targets. `AcTrak`'s control algorithm is driven by two objectives: (A) if a target has not been visited recently, the highest priority should be given to that target, and (B) try to maximize the number of targets that can be seen with the same camera beam or field of view (this is dictated by the camera's orientation and targets' proximity to each other). The second objective saves time via visits to cover multiple targets (instead of multiple different visits to cover those targets). An auxiliary benefit not explicitly considered is that it can detect target group interactions (e.g., when multiple people meet as a group).

Scheduling visitations across multiple targets have to be carefully orchestrated in order to minimize the overall latency associated with changing the PTZ configuration to switch targets; recall that this is a latency expensive operation. For example in Figures 3.2-A and 3.2-B, there are two distant clusters, each containing targets in close proximity. If

the camera alternates between targets from the two clusters (as in Figure 3.2-A), the overall time for making a visit to all targets will be large due to wasteful, repeatedly large PTZ configuration changes. However, if it visits the targets within one cluster before it moves to the other cluster (as in Figure 3.2-B), this overall time will be much smaller (small changes to the PTZ configuration at each step). *Thus, the proximity of targets has to considered while zooming in on different targets.*

Continuing with the example, assume that there are only two targets within a cluster and there is a time budget λ s, associated with visiting these two targets. The camera can stay with the first target for $\lambda/2$ s, and move on to the second target and stay with the latter for remaining $\lambda/2$ s. A better strategy is to alternate between the targets, staying with each for a small period each time. In the first strategy, the camera stays with a target for a longer time collecting continuous (and more) high resolution shots; however these shots are unlikely to yield new information and the other target is not monitored for a big time gap ($\lambda/2$ s). In the second strategy, fewer high resolution shots of each target are obtained due to additional PTZ changes, but the time gap for which a target is not monitored is smaller. Further, each visit is more more likely to capture new activity (e.g., due to a new posture). As discussed later, our framework can be tuned to trade-off between the two strategies.

Overview. As discussed earlier, we seek to balance two tasks (a) tuning the frequency of coverage tours and (b) managing zoom-ins across the multiple targets in the scene. `AcTrak` seeks to avoids wasteful PTZ configuration changes to the extent possible (i.e., zooming out when no new targets have arrived and zooming-in on obsolete locations

of current targets which have moved). RL has been shown to be effective in applications needing adaptive parameter tuning (e.g., [263]), which is what we need to achieve the balance we seek. Our trained RL agent determines the next visit while accounting for the impact of this decision on future decisions, via a simple and quick feed forward operation, depending on the current state of the environment (determined by for example, clustering of targets, application needs, etc.).

To learn the appropriate frequency of coverage tours we monitor: (1) *target arrivals*: if no target appears within the current period between coverage tours, the camera learns to decrease the frequency of coverage tours and (2) *new target locations*: if new targets have moved significantly from the coverage boundaries (have not been detected for a while), the camera infers that it needs to increase the coverage tour frequency. With regards to managing zoom ins, we seek to efficiently determine the next visit while accounting for the impact of this decision on future decisions. A trained RL agent determines this via a simple and quick feed forward operation, depending on the current state of the environment.

In an evolving scene, given what is observed by the camera beam, `AcTrak`'s control algorithm seeks to determine a PTZ configuration that maximizes a given utility that captures our goal. Then, the camera changes its current PTZ configuration to that new PTZ configuration. The PTZ configuration is adapted over a sequence of steps (time epochs) denoted as $k \in [1, \infty)$. We can model this problem as a Markov decision process (MDP), wherein we seek to make a decision with regards to an action relating to a PTZ configuration change, towards either a coverage tour or a specific zoom in, based on the utility. This utility is the long term reward (discussed later) accrued by a Q-learning agent. In an offline

phase, the agent learns a camera control policy that is to be deployed in the online phase to maximize the given utility. Our approach is consistent with previous work [141,215,258]. While alternatively, the problem can be formulated as a semi-MDP that captures steps (i.e., camera PTZ change) of different lengths corresponding to different distances of movement [233], we choose the MDP formulation due to its simplicity and success with previous work (where steps of varying length are also present [141]).

3.2.2 Problem Formulation

`AcTrak` is geared towards a single camera system monitoring a dynamic scene with dynamic target arrivals and departures. As discussed, it considers discrete time epochs $k \in [1, \infty)$ over which it continously adapts the PTZ configuration based on scene evolution. It tracks targets that exist in the scene and saves them in a list. We note that this list may not exactly reflect the ground truth because the camera may only have a partial view of the scene at several of these steps. At each step k, new targets may arrive and may be observed by the camera; we denote those targets as $\mathcal{O}_k$ and the size of the set of these new objects as O_k. Note that it is possible for the camera to find new targets in a zoom-in view (e.g., while zooming in on a door to observe a target that is leaving, a new target may arrive). We denote a set of targets that exist in the scene at step k as $\mathcal{N}_k$; this set includes targets that previously existed at step $k-1$ and O_k; formally, $\mathcal{N}_k \leftarrow \mathcal{N}_{k-1} \cup \mathcal{O}_k$. If the action at step k is a coverage tour, we may remove targets $\in \mathcal{N}_{k-1}$ that do not appear in the scene (i.e., those targets that have left the scene). The maximum number of targets that can exist in the scene is assumed to be N. The key notations is shown in Table 3.1.

We collect features about existing targets (e.g., locations, time stamps at which they were last observed) which we use to compose the state in our MDP formulation (discussed next).

State. The state s_k describes the environment at the k^{th} step of the evolving scene and is defined by the following extracted features from the environment:

1. *The camera PTZ at step k*; the Pan and tilt ranges are $(0, 360)^\circ$, and $(-90, 90)^\circ$, respectively and the zoom ranges from 0 to $m\times$. We denote camera's zoom magnification level at step k as C_k.

2. *Location vectors*: We add three vectors (each of size N) to describe the targets' locations in terms of their associated PTZ configurations: (a) a vector with the most recently updated locations of all targets at step k, (b) the penultimate location where each target was observed and (c) the *location base*, which is the location of a target at which the system was last "positively rewarded" (rewards are discussed later) for observing that target.

3. *Time vectors*: We add three vectors (each of size N) that capture the following time features for each target. Specifically, we add the time difference between the time instance at step k and (a), (b) and (c), respectively: (a) the times when the targets were last observed in a zoom-in mode, (b) the times when the camera acquired the penultimate location for each target and, (c) the time when the agent got a positive reward for each target.

4. *Coverage tour latency*: We also add a feature that includes the time difference between the time recorded at the k^{th} step and the time of the last coverage tour.

5. *Number of visitations:* Finally, we include a vector of size of N that contains the number of visitations that the camera has made to each of the targets up to step k.

Action. An action a_k is performed at each state and leads to a new PTZ configuration which causes a new state to be composed. There are a discrete set of $N+1$ possible actions. These correspond to visiting a specific target (recall that the maximum number of targets is N), or a coverage tour to determine if new targets have arrived.

Rewards. Upon transiting from a state s_k to state s_{k+1} via action a_k, the agent accrues an immediate reward, r_k. The *projected cumulative reward* at step, k, is computed as:

$$R = r_k + \sum_{c=1}^{\infty} \gamma^c r_{k+c} \tag{3.1}$$

where, $\gamma \in [0, 1)$ is a discount factor for the future rewards.

Policy. The policy π selects an action (the next PTZ configuration) at each given state that maximizes the projected cumulative reward for current and future actions. This is defined formally as follows.

$$\pi(s_k) = \arg\max_a \mathbb{E}[R|s_k, a, \pi] \tag{3.2}$$

where a is the action selected from the action space and $\mathbb{E}[R]$ is the expected value of the cumulative reward.

3.2.3 Design of the immediate reward

We define the immediate reward, r_k, when an action a_k is taken in state s_k to be:

$$r_k = PR_k - NR_k \tag{3.3}$$

In the above, we decompose the immediate reward into two parts, a positive reward (PR) and a negative reward (NR), that together drive the trade-off between zooming in and out.

Positive rewards (PR_k): The agent gets rewarded if it discovers new targets regardless of the camera's zoom (this is captured in the first term in the equation below) or successfully revisits an existing target in a zoom in mode if certain conditions are satisfied (captured in the second term in the equation). Formally, this is expressed as:

$$PR_k = \sum_{j=1}^{O_k} \alpha_1 \mathbf{1}(T_j \notin \mathcal{N}_{k-1}) + \rho^{V_j^k} \alpha_2 \{\mathbf{1}(T_j \in \mathcal{N}_{k-1}) \mathbf{1}(C_k \geq Z_j) \mathbf{1}(t - t_j^{base} \geq \tau, |l_j - l_j^{base}| \geq d)\} \tag{3.4}$$

where, T_j represent the targets captured with the camera's PTZ at the k^{th} step. $\mathbf{1}(.)$ is an indicator function that is equal to 1 if at least one of its argument conditions holds true, and 0 otherwise. We capture the prior existence of a target in the scene with the indicator $\mathbf{1}(T_j \notin \mathcal{N}_{k-1}) = 1 - \mathbf{1}(T_j \in \mathcal{N}_{k-1})$ as follows:

$$\mathbf{1}(T_j \notin \mathcal{N}_{k-1}) = \mathbf{1}(\min_{i \in \mathcal{N}_{k-1}} [|v(T_j) - v(T_i)|] \leq \sigma) \tag{3.5}$$

where, $v(T_j)$ represent the visual features of target T_j and σ is a pre-defined threshold; the target was in the scene previously if the visual features match those of a target seen at step $k-1$.

A reward of $\alpha_1 \in [0, 1]$ is accrued if the target is new ($T_j \notin \mathcal{N}_{k-1}$), regardless of the zoom configuration of the camera. A reward of $\rho^{V_j^k} \alpha_2$ is obtained when the camera zooms in on a target but with different conditions (discussed next); ρ is a factor $\in (0, 1]$ and V_j^k is the number of zoom in visits that are made by the camera to obtain the fine grained features of target T_j. This means that for each repeated visit to the same target,

(if $\rho < 1$) the camera receives a lower reward than in its previous visits. This ensures that the camera visits other targets with fewer high resolution images, rather than revisiting a target repeatedly. The reward is accrued if the conditions 1, 2 and at least one of 3(a) or 3(b) are satisfied:

1. The observed target has been seen in the prior step i.e., ($T_j \in \mathcal{N}_{k-1}$).

2. C_k, is larger than the required magnification level for obtaining the fine grained features of the target T_j, (referred to as Z_j as discussed in § 3.2.1).

3. (a) The time gap $(t - t_j^{base})$, between two consequent zoomed in visitations is larger than a threshold, τ, where t_j^{base} is the last time the camera is rewarded when it zoomed in on the target and t is the time instance at the k^{th} step. Note that we make the assertion $t_j^{base} = t$ when the camera visits a target and gets a reward of $\rho^{V_j^k}\alpha_2$.

   (b) The target's displacement between two consequent visitations is larger than a threshold, d, where l_j^{base} is the target's location at which the camera was last rewarded when it zoomed in on the target and l is the location of the target at the k-th step. Similarly, We make $l_j^{base} = l_j$ when the camera visits a target and gets a reward of $\rho^{V_j^k}\alpha_2$.

The last condition helps control the time gap between acquiring two consequent images of the same target. For example, if the user is interested in receiving a continuous sequence of a target's images (video), τ can be set to zero. Similarly, if the user is only interested in tracking a target's activities when its location changes by a certain distance, d can be set to that value.

Negative Rewards (NR_k): The positive rewards do not guarantee that targets are left without being visited (the camera can stick with only one target and still get rewarded). Hence, next we introduce negative rewards (penalty) that coerces the agent to visit targets that have not been visited for a while. Specifically, this penalty (NR) increases in proportion to the time the camera was away from the target (not zooming in on the target) and is given by $\beta \sum_{i=1}^{\mathcal{N}_{k-1}} (t - t_i)$, where t_i is the last time when the camera zoomed in on target T_i, and t is the time of the k-th step.

Note that α_1, α_2 and $\beta \in [0,1]$, are *coefficients* that are selected to balance the positive and negative rewards.

3.2.4 Learning the camera control policy

We find the optimal policy π^* that selects the best action in each state towards maximizing the accumulated reward, R. Since the problem is stateful (i.e., the action is selected based on the state), we use a Q-learning approach [262] as a basis, due to its relevance and ease of deployment. The value of $E[R|s_k, a, \pi]$ is denoted as $Q(s_k, a)$, where the Q-value is updated by the following:

$$Q_{k+1}(s_k, a_k) = (1 - \eta) Q_k(s_k, a_k) + \eta (r_k + \gamma \max_a Q_k(s_{k+1}, a)) \tag{3.6}$$

where $\eta \in (0, 1]$ is the learning rate and $Q_k(s_k, a_k)$ is value of the state action pair at step k (the Q-value is learned by updates at each step).

The optimal value of the accumulated reward at step k that is achieved by taking the action a_k, is given by $Q^*(s_k, a_k)$ and is computed using the well known Bellman equation as follows [232]:

$$Q^*(s_k, a_k) = r_k + \gamma \max_a Q^*(s_{k+1}, a) \tag{3.7}$$

Our model has a finite action space but an extremely large state space, since the targets can exist in any location in the scene. If we have L (quantized) locations where the N targets can exist, the complexity of the state space is $O(L^N)$. Thus, creating a table with the Q values of every state action pair combination is prohibitive. Thus, we use a neural network to approximate the Q value for a given state action pair [106]. The neural network is trained based on previously assigned rewards, and predicts the reward (similar to regression) when a new state action pair is encountered.

For reducing complexity and uncertainty we use a Double Deep Q Network (DDQN) [247], which uses two neural networks to provide an approximate Q value for a state action pair; one is used for action selection and the other for action evaluation. We also use the duelling architecture from [260] to enhance the performance of our model. The interested reader can find details in [260] and [247].

We use the mean square error between the output of the neural network and the Q value as the loss function to be minimized. We apply an ϵ greedy exploration policy, where a random action is selected at a given state with probability ϵ while the action that currently maximizes the reward is chosen with probability $1 - \epsilon$; ϵ, the exploration rate, is tuned to balance exploration and exploitation.

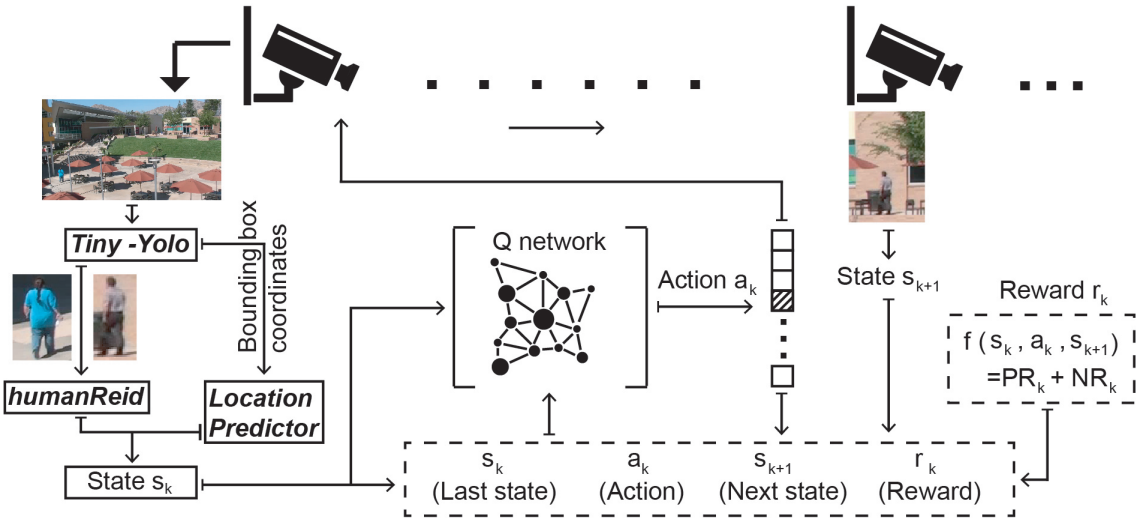

Figure 3.3: A high level depiction of `AcTrak`

3.3 Realizing `AcTrak` in practice

The implementation of our RL framework is on a real camera platform, but a simulator is used for training prior to online deployment. Specifically, we implement our system on Avipas HD PTZ camera (Model: Av- 1080w) [26]. The camera allows pans and tilts with ranges $\pm$ 135 and $\pm$ 35 degrees, respectively. The camera has a 10$\times$ optical zoom capability, quantized over 33 zoom levels. We have developed all our code using Python3 on an Apple Macpro machine, and both the camera and machine are part of wired local network where the machine acquires the frames for processing and controls camera's movement. Our model is capable of adapting to variations in computational, communication and mechanical latencies that could be specific to the system and the setting.

3.3.1 System setup

The camera frames undergo the the following pre-processing steps to compose the RL-state. A pre-trained tiny-Yolo model (YOLOV3 [60]) is used to detect targets in the frames (yielding bounding boxes in terms of frame coordinates) [60, 207]. While tiny-Yolo can detect targets in the processed frame, it cannot determine if the observed targets in the current processed frame have been observed before. This is important for composing the RL state (as discussed in §3.2.2). For that purpose, we crop each target's region from the frame and feed it to the *humanReid* module, which associates the observed target with previously seen targets[2]. Although we obtain the targets' locations in the frame coordinates (using tiny-Yolo), the associated PTZ configuration that makes the target appear in the middle of the frame, with the required zoom level Z_j, is not known. We use the *Location Predictor* module to estimate this. Finally, using the outputs of the two modules, we compute the step rewards and compute the RL-state as discussed in § 3.2.2. A depiction of `AcTrak` is in Figure 3.3.

humanReid:

humanReid (human re-identification module) associates current targets with previously seen targets. Human re-identification is very challenging in itself [24, 249], and is beyond the scope of this work. For ease of experimentation, our targets wear *distinctive* colored apparel to enable the use a light weight pre-trained humanReid deep learning ar-

[2]Technically, we can re-train a new model from scratch that performs the two tasks simultaneously (identifying targets from the raw frame and associating identified targets in the current frame with previously observed targets). However, we choose to use off the shelf trained models and architectures for ease of use, given that these existing models serve their intended purpose.

chitecture, mobileNet [108]. We further tune the model by training it with traces of targets in different colors viz., red, blue, green and black. We collect 24 one minute videos where each target walks in an area of interest with these colored shirts, and we randomly sample frames of those traces to train the model.

Location predictor:

The PTZ configuration required by the camera to observe the target in a zoomed in view cannot be accurately obtained from the bounding box dimensions in frame coordinates. This is because the depth relating to the target (how far is the target from the camera) is not known. The depth can be obtained using monocular depth estimation approaches [32,62,275] that estimate the depth of each pixel in the captured frame; however associating the depth information with the required PTZ configuration to zoom on targets given the camera's intrinsic parameters such as focal length and distortion is not straight forward [166]. A better solution would be camera calibration, a technique via which we can learn the position and the orientation of the camera with respect to the world's 3D coordinates and its intrinsic parameters such as focal length and distortion [277]. However, camera calibration is sophisticated and may not be accurate because of optical lens distortion that may exist in PTZ cameras [120, 264]. Our aim is to make our solution generic and usable by non expert users with a simple pre-setup process.

The approximation we make to support flexibility and generality by trading off accuracy is to estimate the PTZ configuration for an object based on the size of the detected bounding box surrounding that target (detected by tiny-Yolo): the larger the area of the bounding box the more likely the target is closer to the camera. To infer targets locations'

in PTZ coordinates, we use a Decision Tree Regressor [68] that takes as input the camera's current PTZ and the bounding box frame coordinates and outputs the PTZ coordinates for the target. We point out that this approach has sufficient accuracy for our purpose but it is not extremely accurate for two reasons. First, the size of the bounding box is sensitive to the size of the person (e.g., children have smaller bounding boxes than adults even when their depth in the scene is similar). Second, if the size of the bounding box is not accurately determined (due to mis-detection from tiny-Yolo), the estimated location is also affected. As will be discussed, in our real camera and simulation experiments, we were not affected by such mis-detections that may be have been caused by tiny-yolo. Note that estimating the depth is orthogonal to our contributions and our method (i.e., controlling a camera using reinforcement learning) can work in scenes where camera calibration is computed; in other words, it can be combined with a camera calibration method.

3.3.2 Simulator

Creating a large number of training instances with a PTZ camera where each mechanical movement induces high delay (up to 3s in our camera), is inconvenient and prohibitive. For example, training our agent with $\approx$ 800K+ camera movements requires more than 18 days (assuming many targets and training round the clock and that each movement takes 2 seconds). In areas with sparse pedestrians, training will take much longer. Thus, we build a simulator that mimics targets and camera movements to accelerate the learning process. With our simulator, the training takes less than 18 hours as is the norm in such experiments.

Composing training datasets

We evaluate our model via simulations with public datasets, and real camera experiments for which we collect a training dataset (discussed in § 3.4).

Public datasets: We use two datsets (described later) with ground truths of targets' locations (described by bounding box coordinates). This obviates the need for the object detection model or the humanReid model; our *Location Predictor* module uses the ground truths of targets' locations to associate them with PTZ configurations. When zooming in on a target in a dataset, we pick an image patch that coincides with the bounding box of the target. We associate the PTZ configuration latencies associated with moving from the acquisition of one image patch to another. We acknowledge that there is an implicit assumption that the cameras capturing those videos are similar to ours (similar overall delays exist), but since both our platform and these are based on a real world camera, we believe that this assumption is realistic.

Real camera dataset: We collect training videos from the area of interest with the maximum zoom out and label the ground truth target locations for all frames, which can be obtained using the models from tiny-yolo, *humanReid* and *Location Predictor*.

Accounting for `AcTrak`'s mechanical and computational delays

Accounting for the delays experienced in live experiments while training the agent in the simulator is important to ensure that the simulator emulates reality. Otherwise the discrepancy between the real system and the simulator leads to poor performance during

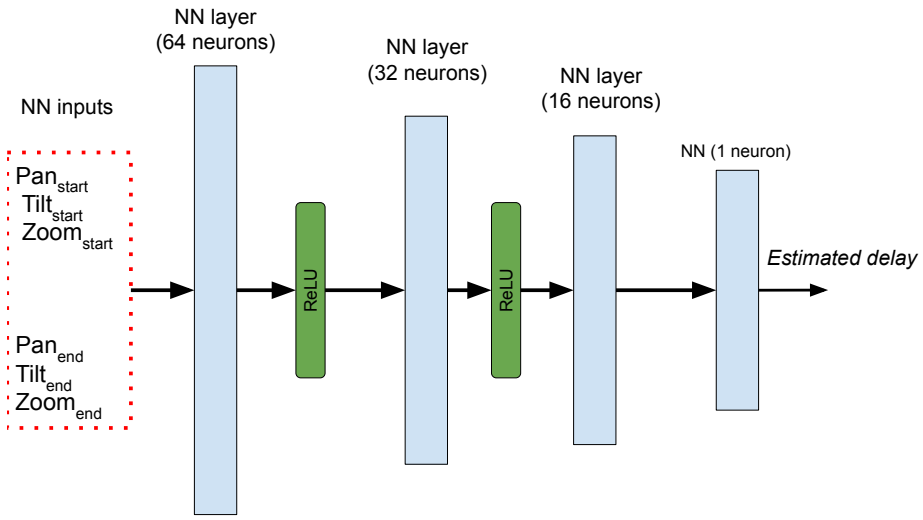

Figure 3.4: The neural network architecture of the *Delay Estimator* model

run time (test time).

Modeling camera's mechanical delays: The time it takes the camera to change its PTZ configuration, depends on the displacement magnitude and the speeds of the camera's mechanical control motors across its axes. We observe that the camera motor does not move with a constant speed, but there is an acceleration component that makes the prediction of the time taken hard. We also observe that the camera's average speed along an axis (e.g., the pan axis) changes even with the same displacement magnitude, if there is simultaneous movement on another axis (e.g., tilt). Besides mechanical delays, there are other factors such as camera response times and network communication latency that contribute to the overall latency. To deal with these challenges, we use a machine learning predictor that takes as an input the starting and ending PTZ configurations, and predicts the time it takes for the displacement (after accounting for all sources of latency). We have collected delay traces from random changes in the PTZ configurations and used

these to train the neural network. We use a simple 3 multilayer perceptron model (shown in Figure 3.4) and a loss function that captures the normalized difference between the true and estimated latencies; more formally, the loss function is $|\frac{time-\overline{time}}{time}|$. Our model has a mean normalized absolute error of 4% and a mean absolute error of 40 ms. We denote this model as *Delay Estimator*.

Modeling computational delays: In real world experiments, we run a *tiny-yolo* module to detect targets in the scene. To emulate edge device computations, we run all computations on a standard computer (with no GPU). Because of that and due to the large frame size of the HD camera, we observe that the computational latency of tiny-yolo is very large (up to 3s in some cases), which causes significant latency. To account for this delay, we run 1000 tiny-yolo queries and record their response times. The response time of 95% of the queries are between 1.8s and 2.5s. We use the average (2.15s) and add a random value in the range of $\pm$ 0.35s when processing each frame, so that the model can account for computational latencies that occur in real deployments.

Modeling camera focus delays: We also experience a delay during the process of the camera focusing on a target. Specifically, we observe that images collected even when the camera has slight motion, are blurry and cannot identify observed targets. Hence, we stabilize the camera at its selected location for 0.2s to ensure that no mechanical noises affect the quality of captured images. We account for this latency in the training as well.

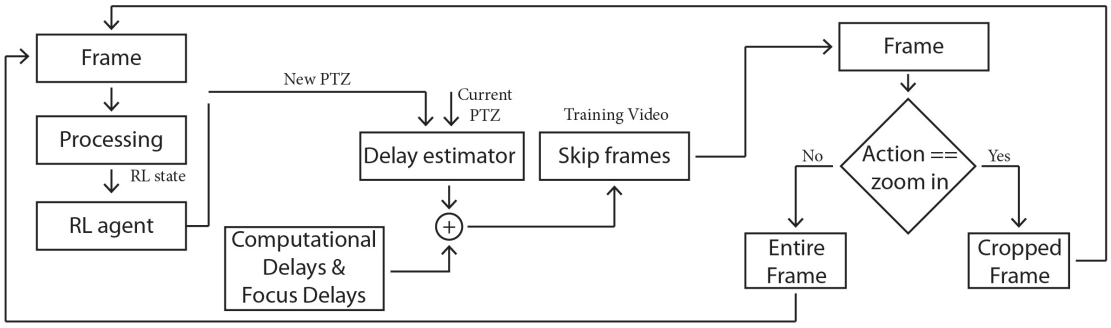

Figure 3.5: A given frame is processed to compose the state as described in §3.3.1, upon which the the agent selects an action (selecting the new PTZ). We compute the latencies associating with PTZ change. Subsequently, we skip a number of frames that correspond to the computed latency and use the first retrieved frame. If the selected action is zoom-in, a cropped frame is passed to the frame processor. Otherwise, the entire frame is passed. .

Training the RL agent

We train `AcTrak` with multiple episodes where each episode is a training video that is made available to the simulator. A flowchart of the training process is shown in Figure 3.5. The frame is processed to compose the RL state that is made available to the agent. The agent then selects an action either based on its learned policy or by randomly selecting an action from the action space depending on the value of ϵ (RL exploration rate). We assume that the camera does not acquire images of the scene while it is processing a frame and changing its PTZ configuration. For computing the mechanical latencies, the new PTZ associated with the selection of the action is now provided, along with the current PTZ, to the *Delay Estimator*, which returns the estimated time (latency incurred) that the camera would need to make such a move. To account for the delays, we skip a number of frames that correspond to the latency obtained from *Delay Estimator* and other latencies, and feed the first retrieved frame to to the object detection system for the subsequent processing.

If the selected action is a zoom out, then the entire frame is considered and the agent can observe all the targets in the scene and record their updated locations. With a zoom in, a cropped image of the frame that makes "the ratio of the size of the bounding box surrounding the target to the cropped image" equal to a pre-selected value, denoted as M, and centered around the target's last observed location, is provided to the object detection system. If the system detects a target(s), it compares its features with those in the existing list of targets, and if there is a match, the system updates the target's most recent location. Otherwise, it adds the newly identified target(s) to the list of the existing targets. Next, the system computes the rewards as described in Eqn. 3.3. A new state is created with the most recent locations of the targets, and the system continues in the same fashion. During test time, we set the exploration rate to zero so that the agent relies solely on its learned policy.

Differences between the trained agent with the live camera and that with the public datasets: When processing public datasets, both in training and testing, we exclude the computational and camera focus latencies so that we can observe the performance under the effect of the mechanical latencies solely. In real camera experiments, we used all the associated latencies while training the agent so that it can work in live experiments and showcase the performance in this realistic setup.

3.4 Evaluations

In this section, we present results from simulations, and from real experiments on our camera platform, to showcase the effectiveness of `AcTrak`.

Baselines: We consider the following baselines:

1. *Standard tour*: We propose this simple baseline that makes the camera *greedily* hover only across the hot spots (areas of interest). We select four hot spots (the most heavily populated locations) using the method proposed in 1001[120]. Note that we did an exhaustive search to find the locations which have the highest populations per unit time; we also find via a search of a plurality of dwelling times that the best value for this parameter is 1 s.

2. *Panoptes* [120]: This is similar to the Standard tour but uses a machine learning model to predict targets' mobility. The model takes as input a target's location and predicts its new hotspot location after a "look ahead" time period. If Panoptes predicts a new hotspot location for a target, it re-schedules the camera tour accordingly (details in [120]). The look ahead times depend on the hotspot locations, and are computed by measuring the average time targets in the dataset take, to move from one hotspot location to another. The mechanical delays for the camera to switch from one location to the other are accounted for while computing the look ahead time periods.

3. *Tracking greedy (greedyB)*: Inspired by Panoptes and Standard tour, this is a greedy algorithm which, instead of hovering over hot spot locations, sequentially hovers over targets' last observed locations and zooms out to discover if new targets have arrived. This baseline uses the same system setup and pre-processing modules as that of `AcTrak` (see § 3.3). We add rule based enhancements to the previous baselines for better performance i.e., if the camera views a target with an incorrect zoom, it zooms in further for the higher resolution image. Since the targets can appear anywhere and

not just at hotspots, and move arbitrarily, we do not try to perform look ahead predictions; we show that greedyB already outperforms Panoptes later.

Metrics: We consider the following evaluation metrics.

- *Time gap between a target's entry and its discovery (TG)*: The difference between the time the camera first observes a target and the time of the target's entry to the scene.
- *Percentage of unseen targets (UT)*: The fraction of targets that enter the scene but leave unseen by the camera.
- *Percentage of targets with zero zooms-ins (ZZI)*: The fraction of targets observed by the camera via a zoom out but for which, the camera never acquires a zoom-in image.
- *Number of visits per target (NumV)*: The number of high resolution shots acquired for each target normalized by the total time the target spends in the scene (# of visits/ second).
- *Maximum time away from target (MTA)*: The maximum gap for which the target's activity was not monitored (i.e., the maximum time between any two consecutive visits to the same target or the time between when the last high resolution target image was acquired and when the target left the scene).

Model: Our model is implemented using Keras (TensorFlow based platform) and it is trained on a Tesla 100 GPU. As discussed in § 3.2.2, we use the DDQN approach [247], where two neural networks with identical architectures but with different weights (updated while learning), are used to compute the Q-values of the state action pairs. The neural network architecture is described in Figure 3.6. We set the discount factor, γ, to be 0.975

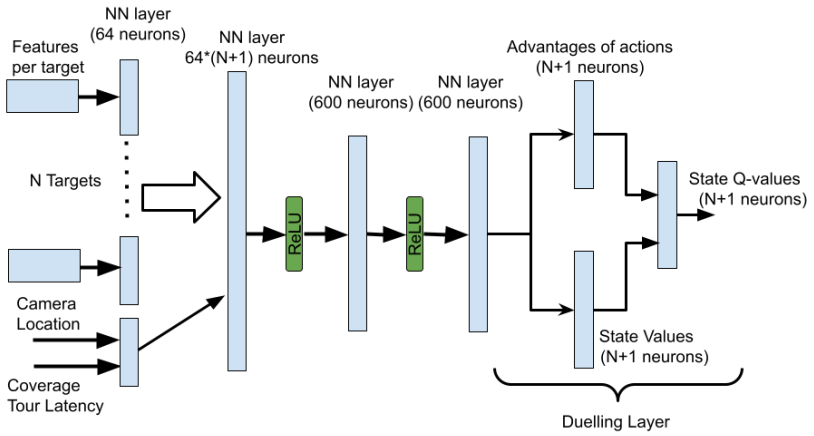

Figure 3.6: `AcTrak` Model Architecture: The neural network consists of multiple layers as shown. First, for each target, a vector composed of its collected features (i.e., features related to timeliness, location and number of visitations as described in §3.2.2) is fed to a NN layer of size 64. The camera location and time coverage tour latency (both features are part of the state) are concatenated and fed to an NN layer of size 64. Subsequently, the outputs are concatenated into a layer of size 64 * (N+1) with a ReLU activation function. Subsequently, the output is fed to two subsequent neural layers followed by the RL duelling layer.

and the exploration rate, ϵ, decays from 1 to 0 at a rate of 0.00001. The learning rate of the neural network is 0.00001. We use the concept of experience replay [106], where we train the agent on its past experiences. We store the actions, states and rewards from the last 30000 camera transitions in memory, where we sample from to train the agent. We show the learning curves of `AcTrak` in terms of the average accrued rewards as a function of number of steps taken by the agent in Figure 3.7.

3.4.1 Evaluations with datasets

Setup: For our dataset based evaluations we use the Virat [189] and Zara datasets [143]. Zara consists of two videos that are taken from a birds eye view in front of a shopping mall, where most of the targets are seen almost with the same depth with respect to the

Figure 3.7: Average reward accrued by the `AcTrak` agent and *greedyB* (with 10% and 90% confidence intervals) as a function of # of steps taken. This is collected with the Zara dataset.

camera. The duration of the first video is 7 minutes and the second is 6 minutes. The videos have footages of large crowds (148 and 204 pedestrians) with different motion speeds and arbitrary entry and egress times. We use the first video for training and the second for testing. Virat contains 28 videos of various durations. The footages are from a camera looking at the scene from an inclined angle where targets are viewed with different zooms. Experiments with Virat show that `AcTrak` works with different camera view angles and heights. Our model is trained with the 20 shortest videos and tests are on the other 8 longer videos (to evaluate `AcTrak` in long term surveillance).

Targets perform different activities in the scene (e.g., walking, standing at a STOP sign); thus the overall times they spend in the scene vary. Hence, in each dataset, we sort the targets according to their speeds in the scene. We estimate a target's speed by measuring

the target's overall displacement divided by the overall time the target spent in the scene. We report the results on 50 % of the fastest and slowest targets, individually to show the merits of our method (they are denoted as *fast targets* and *slow targets*, respectively).

Tuning reward function coefficients: The coefficients are selected based on synthetically collected validation data from the training instances, where we instrument the targets in the training dataset by changing their arrival times to the scene. We tune the coefficients with the objective of minimizing the number of unseen targets and configure `AcTrak` so as to acquire high resolution single shots (the exact coefficients values are defined within the experiment descriptions corresponding the specific dataset). We also report other results (with other coefficients) where we favour the acquisition of continuous high resolution images of targets (video) at the cost of less frequent coverage tours (shown in Table 3.4).

Baseline comparisons: We mentioned earlier that the *tour* mode incorporated in both Standard tour and Panoptes [120] fail when the goal is to track activities of targets that may span the entire scene of interest (locations not limited to hotspots). To show this, we evaluate their performance using the Zara dataset and show the results Table 3.2. The two most important metrics that we are interested in are UT and MTA, because the goal is to ensure discovery of targets that appear in the scene (measured by UT), and track their activities frequently with small time gaps (captured by MTA). Clearly Tracking greedy (*greedyB*) outperforms the other two baselines with regards to both these metrics. This is because many targets do not appear in the locations of interest (i.e., the hot spots). Furthermore, many targets may arrive at these hot spots and leave while the camera is pointed at another hot spot. Although Standard tour and Panoptes are superior in other

metrics, the big gap in UT, TG and MTA make them unsuitable in achieving our dual objectives. Since they are outperformed by Tracking Greedy, we exclude them from the following discussions and use Tracking Greedy (with the label *greedyB*).

We note that adding a prediction module to *greedyB*, to predict or account for targets' future locations is difficult because it is hard to know when and where targets could be, in our scenarios. A simple prediction model similar to the one used in Panoptes [120] cannot be applied because, unlike Panoptes, the target can be exist in any location of the scene, which makes the prediction uncertainty high; we have run experiments and observe that it does not make *greedyB* any better (not shown due to space constraints) and even makes it worse many a time. Coming up with a more sophisticated location prediction method in such scenarios is difficult. This is because the problem of what and when to observe data (i.e: targets' locations) in order to maximize a long term objective (e.g., reliable mobile target location prediction) is a hard problem and is referred to as Active sensing [246,274]. While there are existing solutions that tackle similar problems in different contexts, none, to the best of our knowledge, has tackled this problem before. The closest is called PatchDrop [246], and we discuss in (§ 3.6) why it cannot be utilized in our scenario. We point out here that in fact, `AcTrak` is an deep RL based method that does this task implicitly and outperforms all the baselines (as shown next).

Experiments with Zara: In these experiments, we set α_1 and α_2 to be 0.15 and 0.08, respectively. We set τ to be 2s (50 frames), β to be 0.00035 and ρ to be 0.85. We first evaluate `AcTrak` in terms of the cumulative distribution function (CDF) of the time gap between the arrival of targets to the scene and when they are first discovered by the

camera. As shown in Figure 3.8, 90% of the arriving targets are discovered within 2s, while *greedyB* takes around 4s to do so, for both fast and slow targets. The average time gap with `AcTrak` is 1.25s for both slow and fast targets, whereas the *greedyB*'s time gap is 1.76s and 1.82s for slow and fast targets, respectively. This is because `AcTrak` adapts its zooming out frequency with the expected arrival rate leading to fast capture of targets that step into the scene. *On average,* `AcTrak` *detects new targets in the scene 30% faster than greedyB.* As shown in Table 3.2, due to this fast detection of arriving targets, our agent rarely misses targets that arrive to the scene.

From the observed targets, `AcTrak` captures high resolution images of $\approx 92\%$ of the targets while *greedyB* does so for $\approx 84\%$ of the targets. This is because `AcTrak` prioritizes visits to targets that were not visited before, in addition to smart scheduling of its zoom-ins to capture as many targets as possible without wasteful PTZ changes.

In Figure 3.9, we evaluate the maximum time gap between two consecutive zoom-ins on the same target. `AcTrak` has higher gains in the case of fast targets in comparison with the case of slow targets. This is because, with *greedyB* fast targets' locations observed by the camera become outdated faster than those of the slower targets; hence, when the camera visits a target at its previous location, it does not observe the (fast) target (leading to a wasteful zoom-in visit). `AcTrak` avoids wasteful zoom-ins by observing target displacements and adapting target visits accordingly.

We finally evaluate the number of visits for each target as shown in Figure 3.10. Both `AcTrak` and *greedyB* relatively, exhibit the same performance. The key difference between our agent's visits and *greedyB*'s visits is that the agent's visits are distributed over

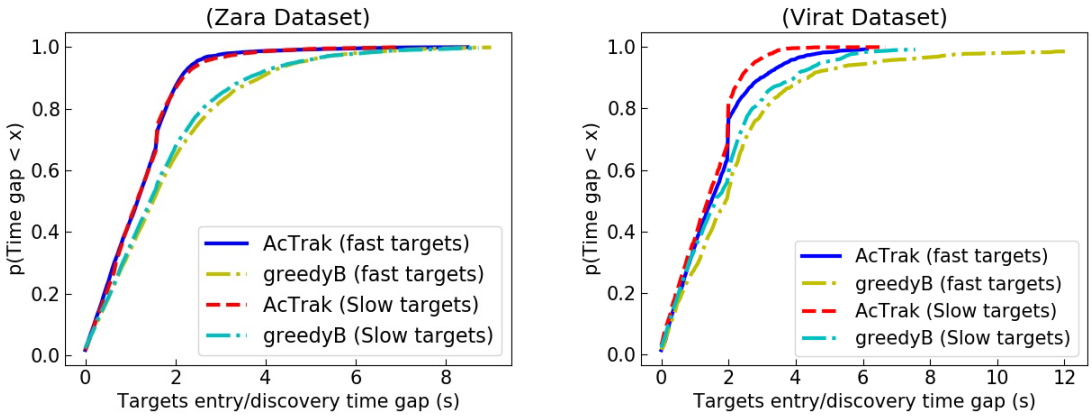

Figure 3.8: CDF of the time gap between target's entry and discovery for Zara (left) and Virat datasets.

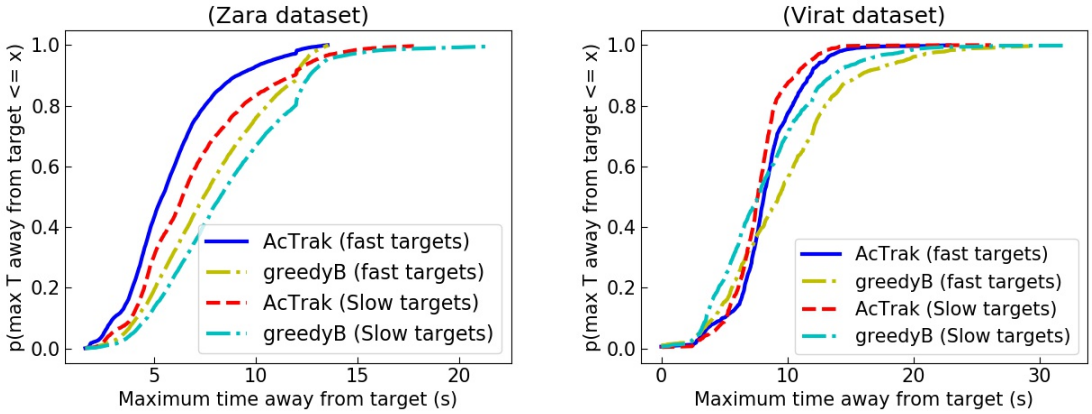

Figure 3.9: CDF of the maximum time away from target for Zara (left) and Virat datasets.

time whereas *greedyB* visits are concentrated over the same time periods leading to big time gaps where certain targets' activities are not monitored (this can be verified from Figure 3.9).

Experiments with Virat: For this dataset, we tune the values of the coefficients α_1, and α_2 to be 0.3, and 0.06, respectively. We set τ to be 4s (100 frames), β to be 0.00025

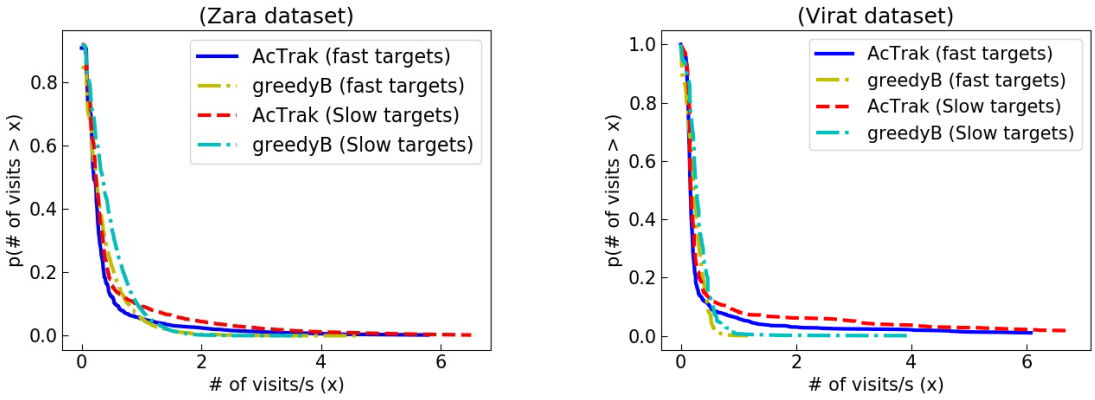

Figure 3.10: Complementary CDF of target's number of visits/s for Zara (left) and Virat datasets.

and ρ to be 0.75. As shown in Table 3.3, the percentage of unseen targets are relatively higher in comparison with the other dataset (but we still outperform *greedyB*). This is because the dataset has mostly videos with random start and end times, and so some targets may appear in the scene in only a very few frames; they appear in the scene right before the video ends or they leave just after the video starts. With the other metrics, we observe a similar trend with this dataset (compared with the Zara dataset) but with lower gains. This is because the camera is positioned with an inclined angle with respect to the scene of interest where targets can move deeper in the scene (their distances to the camera increase) but target's displacement in terms of x and y coordinates (translated to pan and tilt) does not change significantly. Thus, the camera does not lose track of targets easily, as is the case with the Zara dataset (even if the zoom magnification is not met when the camera visits a target, it can potentially know its updated location). The plots associated with the results of this dataset are Figures 3.8, 3.10 and 3.9. We observe that in cases with sparse target arrivals, *greedyB* zooms out unnecessarily wasting opportunities for acquiring

high resolution images. `AcTrak` avoids these and does better in terms number of visits to each target as shown in Figure 3.9. Note that this effect is also seen in our in real camera experiments discussed later.

Understanding the variable τ: In this experiment, we tune the coefficients of the agent's reward function to bias it towards acquiring sequences of high resolution images of targets (videos) rather than single shots as in earlier experiments. Here, our aim is to showcase the merits of `AcTrak` in performing different tasks and its flexibility in tuning the trade-off between zoom ins and coverage tours for different applications. The key parameter that we tune towards this is τ; unlike previously, we now select a small value to cause the agent to acquire a sequence of images (video) of targets. We run again the prior experiment on Zara, but with a different set of reward coefficients. We set α_1 and α_2 to be 0.0 and 0.025, respectively. We set τ to be 0.08s (2 frames), β to be 0.00025 and ρ to be 1. We use the same baseline (*greedyB*) but we vary its dwelling time, so that it can acquire a sequence of images of the same target. With regards to this experiment, we report the average time of continuous video acquired per target, denoted as *TCVA*. The results are in Table 3.4. As shown, `AcTrak` misses 1% of the targets while *greedyB* misses 5.4% of the targets.

3.4.2 `AcTrak` performance with change in "crowdedness" in the scene

In many practical scenarios, the number of targets (referred to as crowdedness [156]) in the scene changes over the day, even in the same scene. We show the performance of `AcTrak` with varying crowdedness. We consider a single scene associated with the Zara

dataset but we create synthetic data where we tune the arrival rates of targets to create varying crowdednesss. In particular, we tune target arrivals rates (i.e., number of targets per minute) in accordance with a Poisson distribution, with mean arrival rates of $\lambda_{small}, \lambda_{medium}$ and λ_{large} per minute; these values are 2, 5 and 10, respectively. Targets' entry and exit locations into and from the scene are selected randomly and the targets speeds are chosen randomly from three different speeds equal to the 25% and 50% and 75% percentile of target speeds in Zara dataset. We evaluate `AcTrak` using the trained model on the Zara dataset on this setup and the results are reported in Table 3.5. As shown, as the arrival rate increases, it becomes harder for the camera to visit targets frequently. `AcTrak` outperforms *greedyB* with respect to all metrics of interest. For example, in a scenario where the target's arrival rate is tuned to be λ_{large} (more frequent arrivals), the percentage of observed targets with zero zoom-ins (ZZI) is 20% when greedyB is used. However, while using `AcTrak`, the ZZI is only 9%.

3.4.3 Real world experiments

We showcase our approach in an IRB approved, real-world setting where volunteers walk randomly in a scene on interest. Figure 3.1 (left) shows the scene at which the real experiments are conducted. To train our agent, we have collected a total of 12 traces of individual random walks (each ranges from 90 seconds to 150 seconds), and we obtain their PTZ coordinates using our *Location Estimator* module. In our simulator (used for training), we further instrument those targets to vary their arrival times over different execution runs, and create many different possible target interactions to enable the agent to learn how to adapt to different scenarios and conditions. For this experiment, we set up the reward

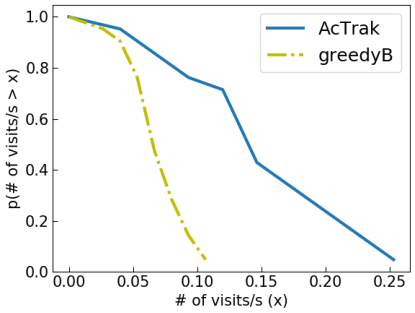

Figure 3.11: Live camera: frequency of visits per target. ($NumV$)

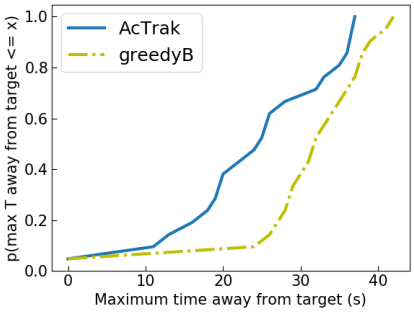

Figure 3.12: Live camera: maximum time camera is away from a target. (MTA)

coefficients (α_1, α_2, β and τ) to be 0.15, 0.1, 0.00001, 8s (200 frames), respectively.

At test time, we have four volunteers that were asked to move randomly in the scene of interest. We select their entry locations and time instances arbitrarily for each new experiment. We have repeated the experiment 5 times using our trained agent, and 5 times using the baseline algorithm). Each experiment lasts for 3 minutes. Our model on average obtains 38 zoomed in shots (per experiment).

Quantitative results. The results from the real camera experiments are shown in Figures 3.11 and 3.12 and Table 3.6. Because of higher PTZ change latencies (associated delays with tiny-Yolo), we observe that the baseline makes many more wasteful moves (zooms on outdated locations of targets or zooms in on a target with a zoom level lower than the required zoom). We also observe that due to the sparsity of arrivals (only four targets over a long period), the baseline zooms out more often than necessary. Our agent outperforms the baseline by a big margin in terms # of acquired high resolution shots. In particular, on average, the agent obtains *2x more high resolution images of targets.*

Figure 3.13: Snapshots from the surveillance videos obtained by `AcTrak` and the baseline (we hide targets faces for privacy reasons). In the first image, a single target is observed via a zoom out view (blue shirt at bottom left near door). `AcTrak` zooms-out much less frequently than *greedyB*, learning that the scenario does not change (no new targets arriving). The second image shows that `AcTrak` while zooming in on the first target, discovers a new target that appears in the scene (light green shirt). The third image shows *greedyB*'s behavior in a similar scenario; it zooms out much more than necessary. There was no other target stepping in to the scene, and instead of zooming in on the red target it zoomed out; this leads to much fewer high resolution images per target, thereby potentially missing activities.

Case studies: Next we present two microscopic case studies of both algorithms from our real deployment (see Figures 3.13 and 3.14).

Unnecessary zoom-outs (Figure 3.13): In this scenario, a single person appears alone in the scene and stays for some long time (around 50s). *greedyB* reacts to this by alternating between zoom ins on the target and zoom outs to continuously check for new targets. However, `AcTrak`, learns upon zooming out and not discovering new targets; it then zooms in on the target for longer times and avoids wasteful zoom outs.

Adapting to high latency (Figure 3.14): We observe that the high latency makes *greedyB* zoom in on empty places because the targets have already left their marked locations (stale data). `AcTrak`'s RL agent learns to tackle this issue and thereby avoids zooming in on outdated locations and performing unnecessary zoom outs. It does so by visiting fast moving targets more frequently, avoiding expensive moves and by opportunistically obtaining updated target locations while it is capturing other targets.

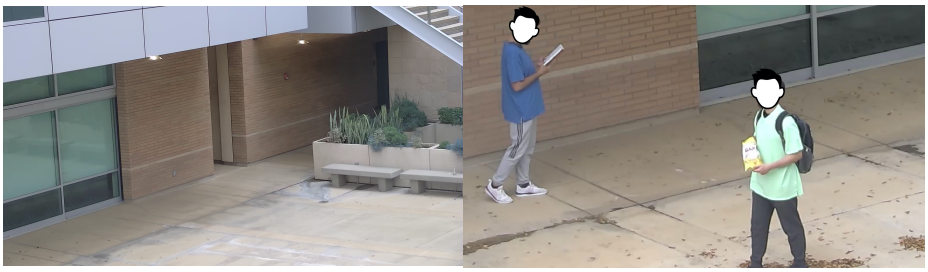

Figure 3.14: Snapshots from the surveillance videos obtained by `AcTrak` and the baseline (we hide targets faces for privacy reasons). The first picture shows a case where *greedyB* goes back to a target and finds the location empty because the target has already left its marked location. This is due to the computational and mechanical latencies and the unorganized patterns of zoom-in on targets (which *greedyB* does not account for). The final image shows that `AcTrak` is able to zoom in on the green shirt target when eating chips and at that point it also re-locates the blue shirt target.

3.5 Discussion:

Single camera vs multicamera: A single camera can address the multiple goals compared to using multiple camers, but provides significant cost reductions. A quick search on Amazon.com [20] and Bhphotovideo.com [33] reveals that a two lens PTZ camera and supporting optical zoom is more than twice the price of a single camera. Thus, our approach can be desirable to small business owners who can just use one standalone camera instead of networking multiple cameras with associated issues such as synchronization and configuration issues, camera calibration issues [225], incompatible software from different vendors and so on. If two cameras are used, one can provide target locations at all times and the other can be used for acquisition of high resolution images. `AcTrak` can be used with the second camera to manage the frequency and patterns of zoom ins on targets. A study of how to harmonize multiple cameras for surveillance is left to future work.

`AcTrak`'s computational overhead: We process the captured frames from the camera

using popular models from computer vision viz., MiniYolo (identifying targets) and human-Reid (associating observed targets with existing targets). Recall that the baseline, *greedyB*, uses the same pre-processing modules that are used by `AcTrak`. The only difference is in executing the trained RL agent that determines what action to take given a state. We run the 10000 random queries using the trained model for Zara dataset on a regular Apple macpro machine, and find that the average response time in terms of determining the proper action, is 4.3ms which is negligible in comparison to the latencies incurred with the computer vision models themselves and the camera's mechanical latencies (which are of the order of seconds). Our expectation is that this time will not increase by three orders of magnitude, even if a less powerful computation machine was used.

Impact of the tiny-yolo's performance on the takeaways: `AcTrak` accounts for latencies due to various factors and this is a key reason why it outperforms *greedyB*. We show that `AcTrak` outperforms greedyB in various simulation and real camera setups where different latencies (e.g., mechanical and computational latencies) are accounted for. In the simulation setup, computational costs are ignored (including those relating to tiny-yolo) and only mechanical latencies are included; this causes the overall latencies to be much smaller than in the case of the real camera setup. This is similar to lowering some of the processing latencies in the real deployment (that might decrease from GPU usage). Based on these results, we do not expect that the take-aways from the real camera experiments will change in flavor, when we run tiny-yolo on GPU or when using a low resolution camera (lower processing delays).

`AcTrak` 's performance when the runtime setup is different from the training setup: In this part, we discuss different forms of 'deviation' between training and test scenarios in the following list:

- `AcTrak` *'s trained model can adapt to crowdedness change:* We observe that the trained model can work even if crowdedness changes over the day with no need to retrain. To showcase this, we run a set of experiments, wherein we vary the arriving rates of targets to the scene of interest. We use the setup associated with the Zara dataset, but we create synthetic data with three different arrival rates. We show that the same exact same trained agent on the Zara dataset can work even if 'crowdedness' changes and can still outperform the baseline. We present these results in Table 3.5.
- `AcTrak` *'s transferability across different scenes:* `AcTrak` cannot transfer to scenes that are not part of the training. For example, the model trained on the Zara dataset may not work on Virat dataset and vice versa. This is because the nature of the scene varies. In the Zara dataset, all targets are viewed from a bird's eye view (e.g., all of them are approximately at the same distance); thus, there is not much variation in the zoom level required to zoom in on targets. In contrast, in the case in Virat, the camera is observing the scene from an angle such that the target distances from the camera vary significantly. Thus, the policy learned by the agent from the training samples from Zara dataset cannot be applied on Virat. In this work, we do not consider transfer learning (i.e., learning a global model with different scenes). This is beyond the scope of this work and requires significant new effort.

- `AcTrak` *'s performance may degrade if target profiles significantly change:* In cases where target profiles significantly change, one can expect the performance of AcTrack to degrade. For example, let us say AcTrack is trained on a dataset from a public park for kids. If the park is repurposed and now teenagers with skateboards and bikes can play in the park, the performance of AcTrack may degrade since the testing data/setup has completely changed from the training setup. In this case, retraining with the new target profiles may be necessary to boost the performance. We argue that in practice such dramatic changes in target profiles are uncommon. In scenes with various target profiles, the training dataset collected from the scene should cover those various profiles. Thus, AcTrack is expected to work. In conclusion, as long as the training video(s) have a good coverage of scenarios and target profiles expected to be present during test time, AcTrack is extremely effective.
- `AcTrak` *can be extended to scenes with dynamic target profiles by using an ensemble of models:* In cases where target profiles and dynamics change dramatically, AcTrack can be simply extended by using an ensemble of trained models, each tuned to the specific dynamics with specific target profiles. During run time, the camera uses the maximum zoom out to observe targets' movements (i.e., targets' profiles) for a small time period. It can then use a neural network which is trained to determine the model that best fits the profile - (it outputs the best model for the specific scenario). For example, consider a public street with walking pedestrians. In the uncommon event of protest/ parade, the model associated with such events can be deployed on the camera. We leave examining this possibility for the future.

Sensitivity of selected reward coefficients: Any RL Framework is sensitive to reward function coefficients (i.e., hyperparamaters) [118, 248]. The rewards need to be tuned with respect to the given setup. We give an example by considering the coefficient τ. To recall, an agent is given a positive reward for visiting the same target if the time gap between two consequent zoom-in is larger than τ. In the Virat dataset, target zoom requirements vary significantly and thus moving from one target to another incurs higher delays in comparison with Zara dataset where all targets have very similar zoom requirements. When selecting a smaller τ (= 2s) similar to Zara dataset), we observe that the camera favors the continuous zoom ins of targets (video) and does not move quickly to different targets. The exercise suggests that using the Virat coefficients blindly, with the Zara dataset or vice versa causes the model to underperform, even to significant extents in some cases. One solution is to use the correct set of hyperparameters with each member of the ensemble from the previous paragraph; in other words, each model in the ensemble has its own set of hyperparameters appropriate for the scenario in which it is to be used. Thus, by using the ensemble, the hyperparameters are also properly changed as scenario dynamics evolve, and thus, can provide superior performance. These aspects are beyond the scope of this current paper, and will be investigated in future work.

3.6 Related Work

There are works that model camera based tracking as a NP-hard, travelling salesman problem [182]. The problem is different from ours in two ways. First, it does not

consider dynamic new target arrivals to the scene (where the camera has to capture and subsequently track them). Second, in our context, targets move arbitrarily with different velocities. Prior work such as [182], impose a pre-determined deadline within which a target has to be visited [182]. This assumption is unrealistic when there is unexpected mobility (the target will be missed). We assume no such deadlines; rather, dynamically changing deadlines are implicitly learnt and the camera avoids wasteful moves (e.g., when it zooms in on an expected target location, it does not find it). Further, `BigEye` induces zoom outs with appropriate frequencies to detect new targets.

There is work in multi-camera networks on co-ordination among the cameras to achieve a particular goal such as target tracking [47, 204, 271] or scene coverage [245] or both [37,65,122,211]. The key difference is that these works assume using multiple cameras to cover the entire scene, thus obviating the need for adaptation to the arrival rate of new targets. However, deploying additional cameras incurs cost, and if certain areas are sparsely populated deploying cameras to cover them always is wasteful (dual optical PTZ cameras are at least twice the price of mono optical PTZ cameras [18]). We consider a more cost effective single camera setting, wherein the frequency of zoom outs to cover the scene fully, are tuned in accordance with target arrival rates while zooming in at other times to acquire high resolution images for activity tracking. We point out that in [65, 122, 211], the focus is on only obtaining high resolution images of existing targets.

There is prior work on using RL to control a PTZ camera's to achieve a single simpler objective. In [27], the authors use RL to opportunistically zoom in on targets that satisfy certain conditions (e.g., a frontal pose available for face recognition). In [135], the

authors use RL to rapidly tune the camera's PTZ to zoom in on a target with a required magnification level from a zoomed out view. However, unlike in `BigEye`, they do not consider the problem of fine-grained tracking of multiple targets nor do they invoke zoom outs to capture new arrivals.

A recent work called PatchDrop formulates an RL approach wherein the goal is to select where and when to acquire high resolution data (patches) to train a model while preserving training accuracy [246]. This work is different from ours in two aspects. First, the work assumes the availability of lower resolution data at all times (the environment is completely observable). In contrast, we assume only a single camera and the environment to be only partially observable while zooming-in. Second, the work does not account for the delays associated with switching between low and high resolution data acquisitions (incurred by PTZ mechanical movements) and between different patches (different targets).

Very few efforts consider a realistic scenario setup like ours viz., the use of a single camera system with multiple objectives [120, 219]. In Panoptes [120], the authors propose a mobility-aware camera scheduling algorithm over a few pre-selected fixed locations (maximizing coverage in these locations only). In contrast, we consider mobile targets whose locations change arbitrarily (not tied to fixed locations). We show in the evaluation section that their approach is not suitable for target activity tracking.

Table 3.1: Key notation

Symbol	**Description**
k	A time step in the discrete set of time steps
$\mathcal{O}_k$	Set of new targets appear in the scene at time k
O_k	# of new targets appear in the scene at time k
$\mathcal{N}_k$	Set of targets exist in the scene at time step k
C_k	Camera's zoom magnification level at step k
Z_j	Zoom level requirement for target j
T_j	Target j
τ	A threshold on the time gap between two consequent visitations for the same target
d	A threshold on the target's displacement two consequent visitations for the same target.
t_j^{base}	The last time the camera is rewarded when it zoomed on target j
l_j^{base}	Target's location at which the camera was last rewarded when it zoomed in on the target j
V_j^k	# of zoom in visits to T_j up to step k
$v(T_j)$	Visual features of target j
γ	Discount on the future rewards
ϵ	Exploration rate to balance exploration exploitation tradeoff

Table 3.2: Baselines evaluations (Zara dataset)

Metric	**TG**	**UT**	**ZZI**	**MTA**
Standard tour	5.24s	59%	0%	98s
Panopets [120]	5.56s	52%	0%	91s
greedyB	1.79s	0.89%	16%	8.1s
`AcTrak`	1.25s	0.09%	8.1%	6.2s

Table 3.3: Virat dataset results.

Methods	**TG**	**UT**	**ZZI**	**MTA**
`AcTrak`	1.49s	1.1%	0.027%	8s
greedyB	2.3s	1.3%	0.11%	9s

Table 3.4: Continuous high resolution shots (video) Results.

Methods	**TG**	**UT**	**ZZI**	**MTA**	**TCVA**
`AcTrak`	1.95s	1%	18%	9s	1.2s
greedyB	3.3s	5.4%	24%	15.8s	1.1s

Table 3.5: Performance with varying crowdedness.

Arrival Rate (λ)	Method	TG	UT	ZZI	MTA
λ_{small}	AcTrack	1.15s	0%	0%	5.2s
	greedyB	1.3s	0%	6%	6.7s
λ_{medium}	AcTrack	1.4s	0%	3%	7.6s
	greedyB	1.9s	0%	11%	9.7s
λ_{large}	AcTrack	2.0s	0.3%	9%	10.1s
	greedyB	2.4s	0.8%	20%	11.5s

Table 3.6: Results of Real world experiments.

Methods	**TG**	**UT**	**ZZI**	**MTA**
`AcTrak`	5	0%	0%	25s
greedyB	7	0%	0%	33s

Chapter 4

DNS Exfiltration Guided by Generative Adversarial Networks

Today, DNS exfiltration attacks are detected by checking for anomalies present in the traffic, such as unusually high transmission rates to a single domain and/or DNS query patterns that are very different from those in benign queries. While such approaches are seemingly robust, we show in this paper that our carefully designed and novel DNS exfiltration attack, Dolos, that uses a generative adversarial network (GAN), can guide the encoding of sensitive data in a manner that both evades these detectors and significantly speeds up the exfiltration rate compared to prior methods. At its core, Dolos divides the exfiltration data into smaller chunks, and projects each chunk into a representation that is very similar to benign queries. In addition, Dolos adaptively tunes its exfiltration rate to conform with benign DNS traffic from the compromised host, and introduces proper levels of spurious traffic to reduce entropy. Importantly, Dolos evades machine learning

(ML) based detectors with no prior knowledge of their architectures or training sets (i.e., it is a blackbox exfiltration). We perform extensive evaluations using multiple datasets and also have a real implementation of Dolos. Our evaluations show that Dolos has a 12% detection probability even if 6 out of the 9 state-of-the-art defenses that we consider, are jointly used to detect exfiltration; if any of today's baseline exfiltration techniques try to achieve the same rate as Dolos in this setting, they are almost surely detected. If we reduce the rates of the baselines to achieve even a low albeit slightly higher detection probability than Dolos (0.15), we see that they take 25× longer to achieve the exfiltration. With the other three defenses, we find that baselines are almost surely detected while Dolos remains relatively unaffected regardless of the rate of exfiltration.

4.1 Introduction

Attempts to steal sensitive information of interest (e.g., credit card details) from compromised hosts is an ongoing goal of attackers [125,234]. One technique for stealing sensitive information is DNS exfiltration [41], wherein adversaries hide and thereby exfiltrate data in DNS queries. Untill a decade ago, DNS exfiltration was not seen as a major threat and thus, enterprises had overlooked inspecting DNS traffic in their intrusion detection systems and firewalls [115]. This seemingly has resulted in an increase in DNS exfiltration incidents [82] and thus, in stolen sensitive data from private networks [125]. In light of this, many enterprises have begun to monitor DNS traffic and have deployed many recent DNS exfiltration defenses [10, 177, 197], bringing about the belief that DNS exfiltration has been effectively curbed. Such defenses mainly rely on recognizing distinctive patterns in

existing/previoius exfiltration traffic compared to benign DNS queries (e.g., entropy of the query, number of capital letters) [10]. In addition, exfiltration detectors monitor traffic to unexpected domains and (1) measure the volume of DNS traffic or/and (2) apply sophisticated information-theoretical approaches to estimate the amount of exfiltrated data that might be potentially embedded in the observed stream [196, 197]. Thus, in the absence of a careful tuning of the exfiltration rate or when simplistic encoding schemes (e.g., Iodine [54]) are used to represent the exfiltration data, these detectors can easily catch exfiltration attempts. *In this work we ask: is DNS exfiltration viable in spite of these defenses?*

Today's exfiltration methods. Existing DNS exfiltration attacks leverage general-purpose encoders (e.g., Base-32/Base-64) to create DNS queries from sensitive data. These methods (agnostic to the type of exfiltration data) transform any arbitrary input data into a specific representation space to comply with DNS rules (e.g., the limited character set allowed in DNS queries) [174]. While they have shown success in the past [125], recently proposed machine learning (ML)-based defenses can differentiate these exfiltration attacks from benign queries with high accuracy [10, 177].

Challenges in the presence of today's defenses. Even if an attacker manages to compromise a host (e.g., in an enterprise via a phishing attack), accomplishing a successful DNS exfiltration attack is not easy. First, an attacker has no knowledge of the defenses deployed by the victim; DNS exfiltration detectors can range from signature-based scanners to much more sophisticated ML-based detectors [10, 48]. Second, the encoding of the exfiltration data must allow exfiltration to occur at reasonably high rates to exploit the data in a timely way. To do so, the encoding must be compact. Beyond this, since detectors often

consider host-specific volumes to detect anomalies, the attacker's malware must determine the proper exfiltration rate that is as high as possible and yet evades detection, with low runtime complexity.

Our approach. In this paper, we design DOLOS (named after the Greek spirit of trickery), a stealthy and efficient black-box DNS exfiltration attack. At its core, DOLOS has an encoding-decoding framework, which is built atop a generative adversarial network (GAN). In brief, by iteratively trying to fool a discriminator neural network (that continuously learns to distinguish between benign and fake queries), the generator learns to map exfiltrated data to a latent space representation which is almost indistinguishable from that of benign DNS queries (and hence, can elude strong state-of-the-art detectors). Because the discriminator is arguably the best detector, refining queries towards evading the discriminator makes the generated encoding extremely effective in blackbox settings (can fool several of today's ML based detectors). Note that formally, a latent space is defined as an abstract, possibly multi-dimensional space that encodes a meaningful internal representation of externally observed inputs. To aid fast exfiltration, the mapping (encoding) is kept as compact as possible, while ensuring that it is decodable with high accuracy at the attacker's external site. Although DOLOS's training uses benign traffic different from that at a compromised host, it learns the intrinsic patterns of benign DNS queries; thus, its outputs online are very similar to such queries even when it is applied to previously unseen exfiltration data. Note that training a deep-learning-based generator on the host itself encumbers high computation cost and requires a lot of training data which is hard to obtain online in a timely way. DOLOS circumvents this issue by training its models offline and porting them onto the victim.

We account for multiple practical constraints, such as composing the exfiltrated data into small chunks that adhere to the specifications of DNS queries [174]. DOLOS also includes a novel rate-tuning module that adjusts the exfiltration rate, guarantees decodability at the remote the site (the encoding itself only provides decodability with high accuracy but no guarantees on its own) and injects appropriate spurious queries based on observed benign traffic from the victim; this prevents the attack from being detected and maximizes the exfiltration efficiency to the extent possible. Put together, DOLOS achieves stealthy, efficient, reliable and stable DNS exfiltration in the wild.

Contributions. A summary of our contributions are:

- We design and prototype a novel generative encoding-decoding framework for stealthy encoding of arbitrary data, efficiently into DNS queries.
- We include a novel exfiltration-rate-tuning module, that includes online mechanisms to ensure proper spurious query injection and reliability in data extraction in conjunction with the above framework, to design DOLOS, a stealthy and efficient DNS exfiltration tool for secretly collecting data from compromised hosts evading several of today's defenses.
- We evaluate DOLOS (with datasets and to a limited extent with a prototype implementation) against 9 state-of-the-art defenses [10,43,48,107,116,154,177,196,197] and compare its performance with traditional exfiltration attack baselines. We find that DOLOS experiences a 12 % detection rate even if 6 of the 9 defenses we consider are jointly used; if the baselines try to achieve the same rate of exfiltration as DOLOS, they are almost surely detected by at least one of the defenses. If their rates are reduced to achieve a 0.15 detection probability (still slightly higher than that with DOLOS), we see that they are

25 × slower than Dolos. With the other three defenses that require to be trained with malicious examples of the attack method, the baselines are almost always detected while Dolos is almost never detected regardless of the rate of exfiltration.

4.2 Background and Threat model

4.2.1 Background

Malware on a compromised host can exfiltrate stolen data by embedding the same in DNS queries. Since DNS resolvers are recursive, such exfiltration queries are delivered to a primary domain of the attacker (e.g., `attacker.com`). An example of DNS exfiltration is shown in Fig. 4.1, where credit card information is extracted from a victim. DNS exfiltration allows opportunistically accessed data to be streamed over a long period without interruption or detection. Importantly, being a critical service, DNS cannot be completely blocked by administrators [13]. In contrast, protocols like ftp and HTTP may be blocked/restricted [52,72]. For example, FramePos, a malware targeting networked Point of Sale (POS) machines, exfiltrated 56M credit card records over six months via DNS queries, after capturing information when cards were processed by the victim POS host [85, 125]. However, state-of-the-art detection methods are effective in detecting and thwarting such attacks [177].

Today's DNS exfiltration attacks. Next, we discuss measures attackers currently take towards successful exfiltration using DNS, while remaining stealthy.

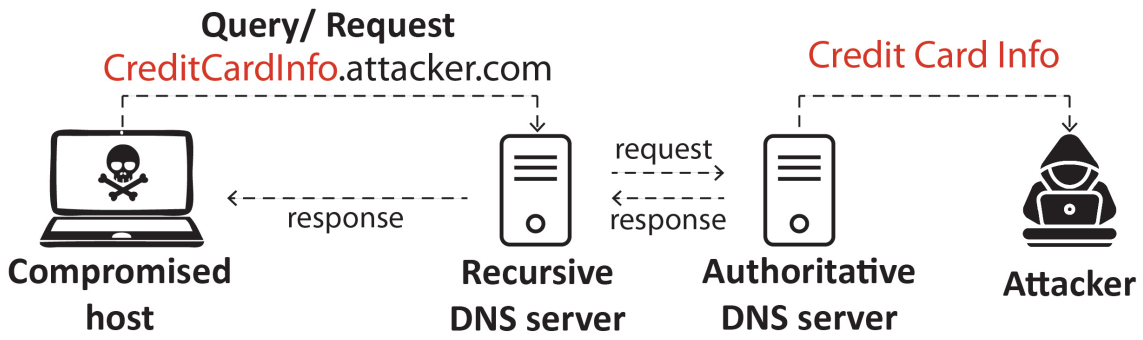

Figure 4.1: An example of DNS exfiltration. An attacker embeds credit card information (in red) in a DNS query destined for its remote domain, "attacker.com". The query is routed to "attacker.com" to resolve the IP address of CreditCardInfo, which enables the attacker to acquire the information.

Acquiring aged domains. If there is a large traffic volume to a *new* domain, many defenses (e.g., a popular one from Palo Alto Networks [180]) trigger an alarm suspecting that the domain was created for DNS exfiltration. To counter, attackers either purchase or compromise aged domains [193].

Choosing common DNS lookup types. DNS supports multiple lookup types [174], the most common ones being `A` and `AAAA`, to resolve IP domains. Other DNS lookup types include `TXT`, used for domain ownership verification and email spam prevention; such lookups carry larger volumes of data [174] and are uncommon. Exfiltration using these latter types is faster, but these types often trigger alerts due to their rarity [105,177]. Thus, attackers typically use `A` and `AAAA`, which limit the rate of exfiltration, but cannot be easily detected.

Bypass caching by choosing small TTLs. Most DNS resolvers cache previously resolved DNS queries to avoid repeated resolutions. DNS responses carry Time-To-Live (TTL) values [174] that dictate how long the resolved query stays valid (in the

cache). Attackers' domains typically respond with very small TTL values to force the DNS resolver to repeatedly resolve the malware's requests to increase the volume of exfiltrated data. Since benign domains also commonly use small TTL values ($\leq 60s$ as per a previous study [197]), detecting exfiltration based on small TTL values is error prone. We point out that in [10], 38% of requests in the studied dataset have TTL values of 0s (no caching). Thus, any method that relies on TTL for classification will cause high false positives. To the best of our knowledge, there is no detection method that uses TTL values to make inferences.

Managing transmission rates. Aggressive transmission of exfiltration queries (at high rates) can be detected even by defenses that simply count requests to a remote domain within short time windows [196]. Hence, attackers use grace periods (e.g., $\approx$ minutes) between queries. One detection method counts the number of cache misses to flag attacks; this implicitly limits the number of exfiltration queries that can be sent in the time window [116]. To compensate for this rate reduction, exfiltration queries can be made longer; however, there are limits on the lengths of DNS queries [174]. Moreover, other defenses can more accurately detect long queries than short ones [123,177,196]. *Thus a challenge in fast exfiltration is how to generate long queries without being detected.*

Encoding exfiltration data. Attackers typically encode exfiltration data for two reasons. First, encoding ensures that the generated query complies with standard DNS protocols. For instance, common DNS request types (i.e., `A` and `AAAA` [174]) only accept 64 characters as the alphabet for body text (i.e., alphanumerical letters, hyphen and dot). Second, it offers some obfuscation aiding stealth. Sending raw data, even if viable, may

trigger defenses that compare embedded DNS traffic with sensitive data (i.e., potential exfiltrated data) from the compromised machine.

To the best of our knowledge, current DNS exfiltration attacks only use general-purpose data encoders (e.g., Base-32/64 and Hex) to map data into a representation space of the characters used in DNS queries [54, 85]. Such encodings however, may differ from benign DNS queries and expose the attack (discussed earlier and in § 4.6.2).

Defensive efforts to detect exfiltration. Previous works assume full knowledge of DNS traffic content (in plaintext) by the detector/defender [10,177,196,197]. We follow the same assumption. While there is increasing encrypted DNS traffic on the public Internet [158], in enterprise environments where DNS exfiltration attacks constitute a major threat, DNS encryption is uncommon [158]; this is because network operators are motivated to monitor DNS traffic and deploy existing defenses to protect the enterprise network [6, 114] .

DNS exfiltration detection. Many legitimate domain names appear to be randomly generated (e.g., "vwdfusdgdkshjdsd.aws.amazon.com") and have become popular [177,197]. Thus, naive defenses relying on the readability of domain names are ineffective. This has motivated smarter defenses that check either the rates at which queries are sent to individual domains or apply machine learning to determine if the features in DNS queries are suspicious. While these defenses are effective in thwarting today's exfiltration attacks, as shown in § 4.6.2, they are ineffective against Dolos.

4.2.2 Threat model

Attack scenario and assumptions. In this work, we consider targeted attacks [266] where the malware acquires and exfiltrates a specific type of data (e.g., credit card numbers

as in the FramePos attack [85]). We assume the attacker has already controlled one or more victim hosts, e.g., via insiders or compromises (this is how exfiltrations happen in the real world) [134, 137]. Similar to the "solarwinds" attack [80, 234], the initial malware file is very small. Subsequently, the DOLOS malware downloads the ML model and necessary files (each of small size) that are used later for exfiltration. Downloading a set of small files to avoid easy detection is commonly used by many advanced malware [80, 234] (users can easily notice large unexplainable files). The malware is assumed to acquire the data either from that machine or from the private network to which the machine is connected (e.g., accessing sensitive infrastructure logs in the private network). The malware can spread to multiple hosts in an enterprise network and all infected hosts engage in exfiltration; this was seen in previous DNS exfiltration attacks (e.g., [137], where roughly 6K devices belonging to the same company were infected).

Similar to FramePos [85], data is assumed to be acquired opportunistically, and the attacker seeks to exfiltrate the data as soon as viable (i.e., timeliness is considered critical for effective use of the data) while evading detection. Fast exfiltration allows quick remuneration. In other words, we assume that the goal of the attacker is fast but stealthy exfiltration.

Defender. Even though the attacker has infiltrated the network, it does not mean that it can exfiltrate the data undetected, as many industrial [13, 114, 209] and research solutions (e.g., [123, 177]) are targeted to stop exfiltration[1]. In practice, such exfiltration detection mechanisms are unknown to the attacker. Upon detecting suspicious primary domains or

[1]Note that exfiltration detection is deployed to catch outbound traffic instead of inbound i.e., our downloaded ML models can still be obtained, hidden as benign HTTP traffic (e.g., with a Trojan Downloader [186]).

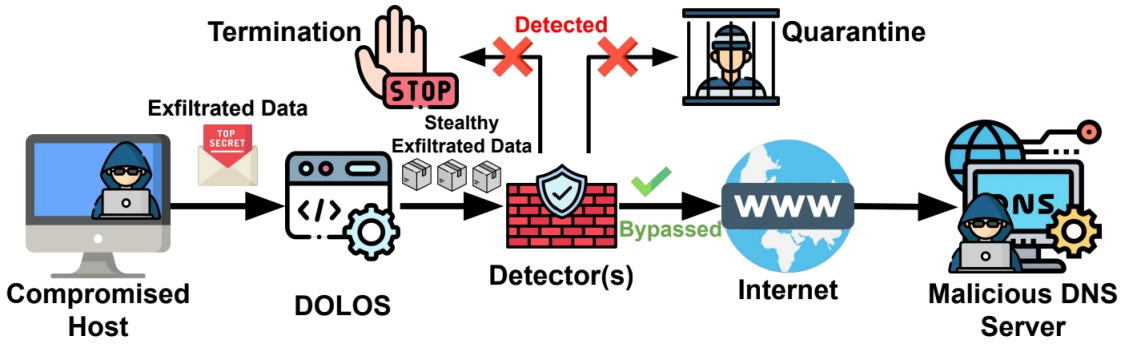

Figure 4.2: Threat Model: employed policies by the defense.

queries, operators can choose one of two strategies to handle them (shown in Fig. 4.2): (1) *quarantine*, which pauses traffic to the suspected domain for a preset period. This strategy suits scenarios that expect higher positive rates from detectors, since it is impractical to manually inspect and verify all suspicious traffic; (2) *termination*, which completely disallows ongoing and future DNS queries to the suspected domain.

Attacker. The malware seeks to steal sensitive data via DNS exfiltration, bypassing an unknown defense using DOLOS. Exfiltration can take place to a single or multiple domains, the later acheiving the full potency of the attack. If the defense uses a *quarantine strategy*, and this is known to DOLOS, it can probe and estimate the best transmission rate that can maximize exfiltration efficiency while avoiding quarantine. Otherwise, DOLOS observes benign traffic on the compromised host, using a sniffer tool (e.g., [223]) to capture the rate of benign DNS requests; DOLOS then tunes the exfiltration rate to be consistent with this rate to avoid detection. We assume that the attacker can attain high privileges on the hosts and mimic benign DNS traffic rates. This is possible via local privilege escalation exploits, which are common [45, 91, 267].

Dolos is trained with samples of exfiltration data offline before infecting the victim. These samples are assumed to be similar to data exfiltrated online. Such samples, for example, for credit card records or computer logs, have well-known formats and can be obtained/synthesized. Similarly, a model trained with an English text dataset can be used for e-mails or other text data, or a model trained with specifiic classes of images (e.g., medical images) can exfiltrate similar images in the wild.

We assume that the attacker has purchased/compromised old domain(s), and uses common DNS query types. Thus, defenses cannot use these to discern exfiltration traffic and must detect the attack based on its encoding and rate only.

4.3 System Overview

We design Dolos to generate embedded DNS queries akin benign traffic; in addition, Dolos includes mechanisms that boost exfiltration rate, while ensuring that the *aggregation* of exfiltration queries remains undetected.

Dolos is based on an efficient encoding method, customized to the data of interest (e.g., credit card records or emails). While prior encoding methods (e.g., Base-64) are generic (no prior knowledge of data is necessary), we argue that using customized encoders for different data types trades off generality (see §4.7) for stealth and speed.

An overview of Dolos's encoder-decoder framework. Dolos's encoder and decoder are trained offline with benign DNS and exfiltration datasets. The encoding ensures that the exfiltrated data representation has high similarity to benign data. It is relatively straightforward to categorize the broad type of networks where exfiltration occurs, e.g., enterprises

(Windows environments, user-facing applications) and data centers (Linux environments, server applications). We can then feed the corresponding types of benign DNS datasets in the offline training phase. We leave the possibility of leveraging a victim's DNS traffic as 'supplemental online training data' as future work. Note that the full training cannot be done on the victim host since it may require a long time, large amounts of training data, and high computational power.

After training, both the encoder and decoder are integrated with the malware which infects the compromised host (reasons for including the decoder are discussed below). The decoder is also used at the attacker's remote site, to which the encoded data is exfiltrated.

An overview of Dolos's online functions. At this point, assume that the malware (equipped with the trained encoder) infects a victim host. Blindly performing exfiltration can still expose the attack because the volume of the aggregated exfiltration queries may not conform with benign volumes generated by the victim. Thus, DOLOS's malware includes a module to sniff the host's benign traffic and tune the exfiltration rate and inject some necessary spurious requests (that are also seen in benign DNS streams), accordingly. A bank of spurious queries is generated offline (consistent with benign traffic) and is shipped with the DOLOS malware, and used during exfiltration. We choose this offline approach since the malware does not have a method to craft spurious queries that are stealthy online; thus online generation may result in anomalies that trigger the detector. In addition, it helps that these offline generated spurious queries can be easily compared with the bank at the external site and discarded.

Finally, note that the encoding generated by Dolos is lossy (although we ensure that the loss rates are very small during training). To fix this issue, Dolos validates the decodability of each exfiltration query with the decoder shipped with the malware. If it is decodable, it is sent as is. If not, Dolos uses an error recovery module (using a traditional lossless compression method in an exterme case) to ensure its decodability. Upon the receipt of a chunk, the remote site uses a simple method (discussed in § 4.3) to apply the proper decoding and recover data. Since such cases are rare, Dolos is still able to evade all considered defenses with very high probability.

4.4 GAN based encoder-decoder design

Next, we describe Dolos's encoder/decoder, trained offline.

4.4.1 Properties of Dolos's encoder/decoder

In this section, we describe the set of desirable properties that guide the design of Dolos's encoder-decoder framework.

Stealthy encoding. Traditional encoding (e.g., Base-64) does not account for stealth, and thus, a steady stream of such outputs are easily detectable by current detectors. To achieve stealth, we need to coerce the encoded exfiltration traffic to resemble benign DNS traffic. While this is challenging, we identify an opportunity to use Generative Adversarial Networks or GANs (details on GANs in [88]) in a novel way towards overcoming it. GANs have been shown to generate examples that mimic a given distribution (e.g., images resembling real humans). However, they have not been previously used to morph DNS exfiltration data. Our key idea is to train a generator to encode exfiltration traffic with the aid of an evolving

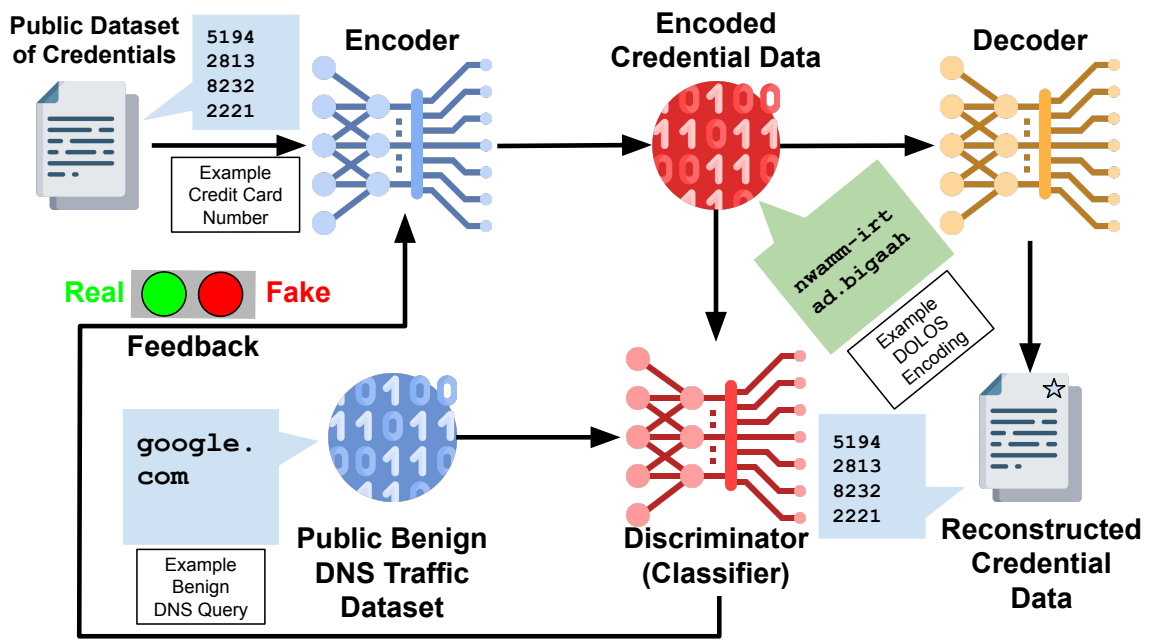

Figure 4.3: Offline training phase of DOLOS. The data from the encoder is constrained to fool a discriminator, and must be decoded by the decoder with high accuracy.

discriminator (trained with benign traffic) that disambiguates such traffic from benign DNS traffic. A well-trained generator then becomes an effective encoder that can transform the exfiltration data into a representation akin to benign DNS traffic. While similar training of a GAN for a single objective (not in the DNS context) has been done in other prior efforts, unfortunately, by itself, this does not suffice. One must also ensure high decoding accuracy at the external site, which is critical for successful exfiltration. Note that fulfilling multiple objectives using the same GAN have been explored to a limited extent in the ML community [14, 51, 273]. However, to the best of our knowledge, the first work to apply this approach to realize a DNS exfiltration attack.

Decoding accuracy. To ensure that the generated codes can be correctly decoded with very high probability, in addition to accomplishing stealth, DOLOS includes a *second* dis-

criminator (we abuse the term here) that is, in effect, an evolving decoder. This decoder is trained jointly with the generator and imposes a second objective to be fulfilled by the latter. Specifically, the encoded representation (a) must deceive the first discriminator and (b) must be translatable to its original form by the decoder. To reiterate, to the best of our knowledge, prior GAN efforts do not consider multiple, different objectives during training.

Code compactness. An encoding that is both stealthy and decodable with high probability, could entail high overhead (lower encoding efficiency). Minimizing this overhead is key for efficient exfiltration. Towards this, we model the problem of finding the most compact encoding as a search problem[2]. Specifically, we begin by considering different levels of compactness (corresponds to different encoding overheads). We use a greedy approach where we try the considered compactness levels in an ascending order (most compact to least). For each, we try to generate an encoding (satisfying stealth and decoding constraints) within a predetermined time period. If unsuccessful, we move to the next. The approach iteratively continues until an encoding is found. More details are provided in § 4.4.4.

Blackbox exfiltration. The discriminator is arguably the best anomaly detector since it learns to discern exfiltration queries as they are iteratively refined to be similar to benign DNS traffic. Thus, if the generation process goes through several rounds of interaction with the discriminator, the encoding is likely to be sufficiently tuned to be similar to benign traffic and can evade blackbox anomaly detectors (as shown in §4.6).

[2]We tried to include a compactness constraint directly in the encoder-decoder formulation, but it increased the time complexity significantly.

Table 4.1: Key notation

Notation	Description
Enc, Dec, Dis	The encoder, decoder and discriminator neural networks, respectively
θ_{Enc}, θ_{Dec}, θ_{Dis}	The parameters of the Enc, Dec and Dis, respectively
L^{m_i}	The length of the exfiltration chunk data
L^{E_i}	The length of the encoding of an exfiltration chunk of data
γ	Ratio of the encoding length of a chunk to the length of an exfiltration chunk
V_D	Validation dataset
$accD$	Validation decoding accuracy
H	Maximum # of batches used for training
B	# of samples in a training batch
α	A weighting hyperparameter to balance the updates from the decoder and discriminator networks.

4.4.2 Encoder and Decoder design

Before delving into the details of our design, we define some notation used in what follows (summarized in Table 4.1). We define random variables that represent the benign traffic and exfiltration data as x and z, respectively. These random variables will have their own distributions in terms of characters in the query, the correlations across the characters, etc. The offline phase relates to jointly training three neural network blocks, viz., an encoder (`Enc`), a decoder (`Dec`) and a discriminator (`Dis`). The parameters of these neural networks (weights) are denoted as θ_{Enc}, θ_{Dec} and θ_{Dis}, respectively. The data to be exfiltrated is divided up into chunks, and each chunk is to be encoded and confined to one *fake* DNS query. We denote a set of chunks as $\boldsymbol{M}$, and each chunk is represented by $m_i \in \boldsymbol{M}$. The encoder, thus, takes a chunk of the exfiltration data of size L^{m_i}, consisting of a sequence of characters $\mathbf{c} = (c^1, c^2 ... c^j ... c^{L^{m_i}})$, and tries to map that on to a codeword $\mathbf{y} = (y^1, y^2 ... y^{L^{E_i}})$ of length L^{E_i}. Note that L^{m_i} may not be equal to L^{E_i}. The mapping function of the encoder is represented by $Enc(c) = f_{\theta_{Enc}}(c)$.

The decoder takes a codeword ($\mathbf{y}$) from the encoder as its input and estimates the original (raw) exfiltrated chunk as a sequence of characters viz., $\hat{\mathbf{c}} = (\hat{c^1}, ... \hat{c^j} .. \hat{c^{L^{m_i}}})$. Given the input $\mathbf{y}$, the decoder function $Dec(\mathbf{y})$ represents the probability that the output $\hat{\mathbf{c}} = \mathbf{c}$, and is denoted by $Dec(\mathbf{y}) = f_{\theta_{Dec}}(\mathbf{y})$.

The discriminator learns how to differentiate between a benign DNS query and a codeword generated by the encoder. Specifically, the discriminator function, $Dis(s) = f_{\theta_{Dis}}(s)$ yields the probability that the given input $\mathbf{s}$, belongs to the distribution of the benign samples. The offline training is depicted in Fig. 4.3.

Stealth. Since the discriminator seeks to differentiate between benign and fake exfiltration queries, it tries to minimize the cross entropy loss between the input and the correct output (which is known as ground truth during training). Let us denote the probability of the discriminator's prediction on the generated queries (fake) and the benign queries as $Dis(Enc(z))$ and $Dis(x)$, respectively; here, z is the exfiltration data fed to the encoder, and x is a benign DNS query. To minimize the cross-entropy as alluded to above, the discriminator will seek to minimize the loss function: $\min[-log(Dis(x)) - log(1 - Dis(Enc(z)))]$. This, in turn, is equivalent to $\max[log(Dis(x)) + log(1 - Dis(Enc(z)))]$.

At the same time, the encoder seeks to fool the discriminator by minimizing the discriminator's confidence (probability) with regards to labeling the generated fake queries. In other words, it wants to minimize $log(1 - Dis(Enc(z)))$.

Given the conflicting objectives of the discriminator and the encoder, we can model their interactions as an iterative minimax game with the following loss function ($\mathbb{E}_x$ and $\mathbb{E}_z$ are the expectations over benign and exfiltration data):

$$\min_{\theta_{Enc}} \max_{\theta_{Dis}} \quad \mathbb{E}_x[log(Dis(x))] + \mathbb{E}_z[log(1 - Dis(Enc(z))]. \tag{4.1}$$

Decodability. To ensure the decodability of the generated codes, we jointly train a decoder. Here, both the encoder and the decoder seek to maximize the probability of correctly predicting the original characters from the latent space encodings. This translates to a minimization of the average cross entropy between the inputs and the ground truth labels. This cross entropy loss minimization is given by:

$$\min_{\theta_{Enc},\theta_{Dec}} \quad \mathbb{E}_z[-log(Dec(Enc(z)))]. \tag{4.2}$$

4.4.3 Practicalities

Neural network architecture. We need a neural network architecture that captures semantic relationships as well as short- and long-term dependencies across the characters in a benign DNS query. If the learnt embeddings reproduce these properties, they can better mimic those queries. There exist many neural network architectures that satisfy the above properties, especially in the NLP space, where capturing semantic relationships is critical. Among those, we choose transformers [251] as our choice since a transformer allows for parallel computations of sequential data, which makes the training fast. One nuance is that, typically, transformers take words as inputs; since we want our approach to work with different types of input data (e.g., credit card numbers, text data), we choose our inputs to be characters instead of words. Note that as discussed in detail later, even more complex data forms (e.g., images, which we consider in this work) can be represented using this method (e.g., with an image, each byte representing pixel intensity can be considered as a character and fed to the model).

Representation of the latent space. DNS queries `A` and `AAAA` permit only 64 characters. Thus, the encoder's output (i.e., $y^1, ..y^{L^{E_i}}$) is a sequence of discrete characters from these.

Non-differentiable discrete latent space. Our inputs are discrete characters, and so are our latent space encodings. Back-propagation, used to tune the neural network weights, cannot be directly applied to discrete variable representations that are non-differentiable (i.e., they have zero gradients) [121]. To overcome this, we use a popular solution for discrete representations, viz., the softmax-Gumbel approximation [121]. The idea is to use discrete variables in the forward pass, but use continuous approximations in the backward pass.

4.4.4 Training algorithm

We train DOLOS to optimize the objectives in Eqns. (1) and (2) using an iterative algorithm. Iterative methods are often used in GANs [88]; however, as discussed, the novel aspect of our work is that we also seek very high likelihood of decodability and compaction.

Towards iteratively optimizing the objectives in equations (1) and (2), we update the weights of the neural networks after each batch of inputs, until we generate stealthy *and* decodable, fake DNS queries. Specifically, we *first* sample a batch from benign DNS traffic and a batch from the output of `Enc` to update the weights of `Dis`. In the *second* iterative step, the same batch from the `Enc` is fed to both the `Dec` and `Dis`, and feedbacks from both are used to update the weights of `Enc`. Since the updates from both networks may vary in magnitude and effect, the encoder may be forced to favor one objective over the other. We use and tune a hyper-parameter α to balance the two objectives. In the *third* step, we update the `Dec` weights to enhance the decoding accuracy. The three steps are repeated until DOLOS is able to successfully bypass a validation step (discussed below in what follows). The offline training of DOLOS is captured in Algorithm 2.

Compactness. As discussed in § 4.4.2, we seek compaction to increase the exfiltration rate. For a given length of a raw chunk, the generator is constrained to output a fixed (to be determined) length encoded query (regardless of the semantic content of the raw chunk). We seek to find a value of $\gamma = \frac{L^{E_i}}{L^{m_i}}$, that allows us to map a raw chunk of length L^{m_i} to the shortest possible encoding length L^{E_i} output by the generator. This would then maximize the efficiency of the encoding (highest amount of information encoded into the smallest number of characters in the latent representation). In other words, we search for the smallest

value of γ, such that the encoded query is decodable, and preserves stealth. Specifically, any γ smaller would violate either stealth or decodability or both. For simplicity, we confine the search space of γ between 0.5 and 1.5 with step 0.1. We begin with the smallest γ (which yields the most compaction), and if the model does not meet the the criteria used to stop training (discussed next), we re-initialize the models and train them with the next larger γ value.

Once γ is thus determined, if we know what is the maximum permissible encoding length L^{E_i} (the maximum length of DNS queries sent by the victim host), we can compute the corresponding raw chunk length that can be used as $L^{M_i} = \frac{L^{E_i}}{\gamma}$. We then collect tokens to fill an m_i smaller than this length and generate the encoding during online operations as discussed in § 4.5.

Validation. Since it is very hard to fool the evolving discriminator (as it continuously learns), we use a validation process to determine when to stop training. After every N batches (1000 in our evaluations), we first test the decodability of the generated codes using the trained decoder to ensure it meets the decoding accuracy constraints. Subsequently, we test the stealthiness of the generated codes against an anomaly detector (not the `Discriminator`) just trained on benign DNS queries. If we fool this detector with a very high probability (> 99 %), we assume that the GAN has been sufficiently trained. We note that the anomaly detector is different from the defenses we test DOLOS against, and thus it does not violate the blackbox assumption.

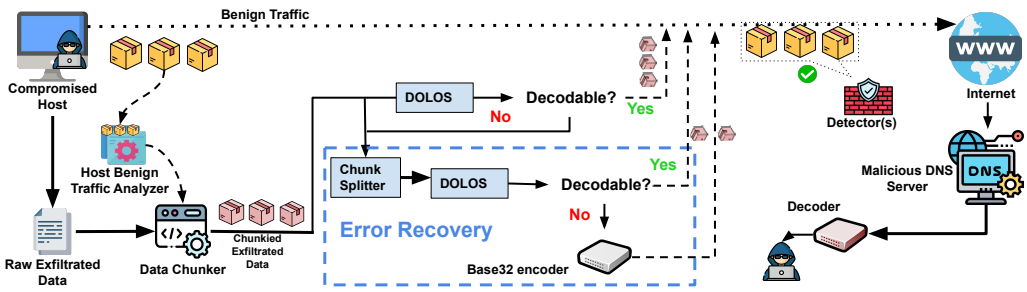

Figure 4.4: Online attack phase of DOLOS. DOLOS sniffs benign traffic to tune the exfiltration rate in terms of number of queries transmittable in a time window and chunk length. Next, it divides the data into chunks and encodes them as DNS queries. If the encoded query is decodable, it is sent as is; else the error recovery module is used.

4.4.5 Composing spurious queries

As discussed in §4.3, we form a bank of spurious queries offline by sampling the generated traffic from batches in a validation dataset and identifing the most frequent 3-4 characters. We randomly combine these along with natural separators present in DNS queries, viz., 'hyphen' and 'dot', to form spurious queries. We refine these with our discriminator until validation.

4.5 Tuning the exfiltration online

Next, we describe DOLOS's operations on an infected host. We reiterate that DOLOS's encoder is unaware of the defense, or the policies employed upon flagging a domain as an attack site.

Most of today's defenses make an inference on queries sent to each primary domain (i.e., decide if that domain is an exfiltration site or not) [177, 196, 197]. Such inferences are based on the volume, the rate, the repetition of queries and the entropy associated with

the aggregation of queries to that domain. To evade detection, DOLOS must tune these parameters for each domain to which it exfiltrates data (can do so independently), towards achieving evasion with respect to those domains.

The best rate for stealthy exfiltration. DOLOS observes benign traffic over an empircally chosen time window (few hours) to estimate the exfiltration rate. In particular, DOLOS needs to choose the number of requests (N), and the average query size, (L), in each time window. The bigger these values, the more data can be exfiltrated, but if they are too large, detection is very likely. A naive approach is to observe the number of requests and the average length of requests to each domain, and from among these, choose the N and L that would maximize the exfiltration rate (i.e., $N * L$ per time window). However, as discussed benign traffic consists of many repetitions of queries (either partial overlaps or full repetitions). To be consistent with benign queries, the attacker has to transmit unique exfiltration requests *and* repeated requests, and requires an estimation of the rate of requests in each category. Note that determining exactly how many times each query is repeated is not necessary as this value differs across different primary domains and we have not seen it being used in practical defenses. In other words, the percentage of unique and repeated queries transmitted to each domain should be consistent with the repetition rate seen in benign queries sent by the victim host.

Beyond repetitions, many requests are partially similar in benign DNS traffic. Not accounting for partial similarity may expose the attack [197]. To illustrate, the following two unique requests are considered partially similar: `gllto1.glpals.com` and, `gllto2.glpals.com`. To evade the detector, DOLOS includes both repetitive and partially

overlapping spurious queries consistent with benign traffic; these are later ignored after exfiltration.

Key idea. To estimate the volume of "unique" or dissimilar queries for each primary domain (obtained from the benign traffic on the compromised host), we cluster the associated queries; those belonging to the same cluster can be deemed similar or repetitions. From these, DOLOS identifies the domain for which the combination of average query length and number of unique queries yields the highest exfiltration rate, and uses these values in tuning its exfiltration process.

Clustering algorithm. Existing clustering methods, including even the simplest of them (i.e., k-means clustering [157]) are expensive. This is because k-means requires multiple iterations of comparisons among the data to converge, and a large space complexity to store all the queries from the host. Importantly, the proper "k" is not known a priori. Because of this, we design a simple algorithm for DOLOS. In brief, for each primary domain, the algorithm processes the streamed queries. With each query, it measures the similarity between the query and the *cluster representatives* of previously formed clusters; if the query is not similar to any representative, a new cluster is created with that query chosen as its representative.

To assess the similarity between two DNS queries, we use the following approach. For each pair of queries, we measure the Jaccard similarity [117] by computing intersection between the characters of the query relative to the length. If this value is greater than a pre-selected threshold, we consider the queries to be partially similar. To select the appropriate threshold, we conduct offline analysis using samples of benign traffic and find

that a threshold value ranging from 0.7 to 0.8 effectively groups similar queries. The algorithm has a $O(n * d)$ run time complexity where n and d refer to the number of DNS queries sent by the host to a primary domain and the number of clusters per primary domain to which queries are sent, within the time window of interest. This process is captured in Algorithm 3.

Online exfiltration. DOLOS's online workflow is shown in Fig. 4.4. DOLOS computes the number of unique queries that it can transmit in a time window, as well as the number of spurious queries it must insert, based on the clustering it has constructed[3]. DOLOS computes the most frequent characters used in the encoded exfiltration queries; it then chooses the spurious queries that are closest to the cluster members (in terms of Hamming distance) from the pre-stored bank (recall § 4.4.5). Subsequently, for the given exfiltrated data, DOLOS encodes chunks of the proper size (it computes the size based on the learned value of γ as discussed in § 4.4.4) to form fake queries using the pre-trained encoder. The exfiltration (fake) and spurious DNS queries are sent in that time window.

Decodability assurance: Since the latent representations generated by DOLOS are inherently lossy in nature, a small subset of chunks may not be decodable (details in § 4.6.2), despite considering decodability during training. To guarantee decodability, DOLOS checks if the encoded chunk is decodable using the downloaded decoder (a replica of that used at the remote site). If the chunk is not decodable, DOLOS uses an error recovery method to guarantee decodability. Our module for error recovery attempts to use DOLOS's encoder but with shorter chunk sizes, and if it fails, DOLOS replaces the encoded chunk

[3]The number of spurious queries is the difference between the total number of queries and the number of unique queries that are determined by our clustering (for each domain).

with a new fake query using Base-32 encoding (which is lossless). We use Base-32 because it performs better as compared to other traditional encoding methods in terms of stealth (e.g., Base-64). As shown in § 4.6.2, the fraction of chunks needing error recovery is very small and the effect on DOLOS's stealth and speed is negligible.

The error recovery model shown in Fig. 4.4 works as follows. Upon finding a chunk with decoding errors, DOLOS's online module shrinks the input chunk size by a factor β (set to 0.8 empirically in our experiments) and attempts to encode the new chunk with DOLOS, again. If this fails, the step is repeated a second time. If the two attempts still cause a decoding failure, DOLOS uses Base-32 for encoding. While such queries can be flagged by defenses, because they are rare, the domain to which data is being exfiltrated are not deemed malicious by defenses (they need to observe a sustained stream of such queries to do so) as seen in § 4.6.2.

Decodability at the remote site: Upon receiving a DNS query, the remote site has to determine whether the received query is encoded using Base-32 or DOLOS's encoder. We use a simple solution for this task. The receiver attempts to decode the query using a Base-32 decoder; if the query is not in the correct format, the Base-32 decoder issues an exception. In that case, the receiver infers that the received query is encoded using DOLOS's encoder. Otherwise (no flag), it deems that the query is encoded using the Base-32 encoder. While it is possible for the Base-32 decoder to decode DOLOS encoded queries in extremely rare occasions, we did not observe those in our experiments. To cope with such cases, one can apply other solutions to make the decoding more robust. For example, the outputs of both decoders can be combined with previously decoded chunks to evaluate which one is

more consistent with the received stream. Another possibility is to use a query classifier (similar to the work in [48]) that discriminates between Base-32 and DOLOS encoded queries.

4.6 Evaluations

We evaluate DOLOS with multiple types of exfiltration data and against multiple defenses, and compare it to prior encoding baselines. We first describe our setup, and then our experiments and results.

4.6.1 Preliminaries

First, we describe the datasets used, our implementation, the encoding baselines, and the considered defenses.

Datasets. We use multiple datasets in our evaluations.

Benign datasets: We use two datasets for training DOLOS (trained on each at a time) and two different others for training the ML based defenses. The use of different datasets for DOLOS and the defenses emulates blackbox settings. DOLOS is trained using:

- Georgia Tech DNS dataset (GT) [235]: This dataset contains DNS traces collected from suspect Windows executables in a sterile, controlled environment; thus, they often do not generate malicious queries. These exceutables often use benign DNS queries to test connectivity [78][4]. We collect PCAP files dating from 2015 until December 2020, and use it to train DOLOS[5]. DOLOS that is trained on this dataset is denoted as DOLOS (GT).

[4]To verify that the dataset contains only benign traffic, we use the best defense in our study [10] (discussed later) on the Georgia Tech dataset, and find that the triggering rate is negligible.

[5]This dataset was used in [78] but was recently withdrawn. We learned from the authors of [78] that this

- ISI-reverse DNS queries (ISI-rdns) [188]: This dataset is collected by using a reverse DNS scan over the entire IPv4 space. While a subset of the IP addresses may lead to names that are not associated with real domains, we try our best to filter these out using heuristics based on our domain expertise. Domains with a fraction of numerical and capital characters, larger than 30% are removed. Moreoever, many queries (e.g., to the same primary domain) are similar, and these can create bias in training DOLOS. To remove these, we cluster queries using our method in §4.5 and only use diverse samples. We denote the version of DOLOS that is trained on ISI-reverse DNS queries as DOLOS (ISI-rdns).

Datasets used for training state-of-the-art defenses are:

- Thapar dataset [222]: This dataset was collected from 4,000 hosts in a university over 10 days. We extract DNS queries from successful DNS responses (e.g., DNS queries associated with NXDOMAIN responses are ignored) [164]. Since the dataset is from an operational setting, our belief is that it represents data used to train real DNS exfiltration defenses.
- ISI host Level dataset [187]: This dataset contains massive raw packets collected at a b-root DNS server with anonymized IP over two days. We group requests by the srcIP field corresponding to the recursive resolver (i.e., each of which is a host). Since this huge dataset is collected from real users, our belief is that a defense trained with dataset should be able to differentiate between benign and exfiltration queries.

was due to funding/maintenance issues.

Exfiltration Datasets: We consider multiple types of exfiltration datasets to evaluate DOLOS. In all cases, we separate the records into training and testing sets. At most a third of the records (selected randomly) are used for training and validation, while the rest are used in testing.

- *Text dataset*: We use the Amazon Reviews Dataset [165] to represent text data (e.g., emails or documents) that the attacker may compromise. We consider the data in terms of characters (not words as discussed in §4.4). This dataset contains 168 unique characters (English alphanumerical characters and special characters).
- *Credit Cards*: We mimic real credit card information to create our own synthetic dataset (2M records). Each synthesized credit card record contains a 16 digit credit card number where the digits have to pass the Luhn algorithm test [160], a method that is used to verify synthesized credit card numbers. In addition, the four digit expiry date, the three digit CVV, the first and last names, the address and billing zip codes are also generated as follows. The CVV is just three random digits and the expiry date is randomly chosen between Jan 2024 and Dec 2034. To generate names, we use the dataset in [208] which contains around two million real names. For the address and billing zip codes, we download US west, midwest, northeast and south addresses from *batch.openaddresses.io* [190].
- *Computer Logs*: Computer logs can be useful for subsequent reconnaissance attacks (e.g., [103, 155]). We use two datasets of logs. The first is a Microsoft Windows "Event Logs" dataset (from a public GitHub repository of logs collected over 226 days [1]) of size 27GB. The second is a Linux logs dataset [102], collected from a Linux server over 264 days.

- *Images*: We use a dataset of x-ray images (in PNG formats) that were used in COVID diagnostics [126]; such data became valuable recently for attackers [210]. We transform the image from the PNG file into a matrix of bytes (each representing pixel intensity). Note that PNG image formats offer lossless compression and the actual values of pixel intensities are retained without change. Thus, this transformation does not result in any loss of data. Further, the matrix is processed to form a sequence of characters/bytes. In other words, we flatten the matrix to a single dimension (sequence of bytes) which is then input to the encoder; the matrix can be reconstructed at the receiver. DOLOS initially transmits the metadata describing the image shape, so that the receiver can reconstruct the same.

Encoding Baselines. We compare DOLOS's encoding against three types of baselines (which we implement):

- *Iodine [54]*: Iodine, the popular DNS exfiltration tool, which compresses data and encodes it with Base-64 (denoted Iodine-64) *or* Base-32 (denoted Iodine-32). While Base-128 is available on Iodine, it does not comply with DNS types `A` and `AAAA`.
- *DNSCat (Compressed HEX) [112]*: DNSCat is a popular DNS exfiltration tool that compresses and encodes the data using Hex-encoding into strings.
- *FramePos encoding* [85]: This encoding was used in the recent attack where credit card information from POS was exfiltrated over DNS. It is essentially a variant of Hex encoding that does not compress data. After encoding the exfiltration data, each byte is XOR-ed with a pre-determined integer value; to decode, the received value is XOR-ed with the same integer at the attack site to retrieve the original value. Details are found in [85].

Defenses. We can categorize defenses into three types: (1) rule based defenses; (2) anomaly detectors (ML based); and (3) classifiers that distinguish between bengin and malicious classes using ML.

Rule based defenses impose empirically derived rules on some properties of outgoing DNS traffic. We summarize rule based defense methods below:

- *Zeek* flags a domain if (i) the length of any transmitted query or (ii) if the number of unique queries within a time window, to the domain exceed preset thresholds.
- *ZeekQ* is similar to Zeek but adds a rule to check if the percentage of numerical characters in a query exceeds a threshold [107].
- *Paxson et al.* [197] collects queries to a domain over a time window and compresses them; if the compressed volume exceeds a certain threshold, the domain is flagged. While Zeek and Paxson el al., issue alerts based on the traffic volume to a domain, they fail if the attacker exfiltrates data to multiple domains with low rates.
- *Ishikura et al.* Unlike the above, [116] builds a shadow least-recently-used (LRU) DNS cache (a copy not interfering with DNS directly), which counts cache misses in a time window. If the number of shadow cache misses for a given client exceeds a threshold (i.e., the maximum number of cache misses across all clients in the prior time window), the defense flags an attack.

Anomaly detectors, listed below, learn features in benign DNS queries or the aggregation of DNS queries to a domain, in a time window. They detect deviations from benign queries and flag existing attacks even if exfiltration is at low rates.

- *Nadler et al.* [177] uses an isolation forest [152] to detect anomalous domains and is

adopted by Akamai [13]. The features used are: average length of queries, number of queries, fraction of unique queries, average length of readable subdomains, the aggregated entropy of all transmitted queries and the fraction of DNS types that are `A` and `AAAA`.

- *Jawad et al.* [10] uses a set of hand-crafted features to build an isolation forest anomaly detector. The features are: query length, # of capital letters and numbers, # of subdomains, average and maximum lengths of subdomains, and the entropy of the request.

Classifiers are trained with both benign and malicious samples (assumed to be known to the defender), and perform classification at a DNS query level. Below we list such defenses.

- *Buczak et al.* [43] uses Random Forest to classify benign and malicious queries. A total of 17 features are used including query shape features such as ratio of distinct characters, maximum and average length of subdomains and percentage of numerical characters.
- *Liu et al* [154] uses Support Vector Machine (SVM) to classify benign and malicious queries. It uses the entropy of the uni-gram, bi-gram and tri-gram of characters as features.
- *Chen et al.* [48] trains an LSTM classifier with samples of benign and malicious traffic.

In all cases we follow the directions on training and tuning the defensive models from the original papers. Since the last three defensive models need adversarial samples to train, for each attack method, we provide examples generated by the same method (e.g., the classifier is given Iodine-32 examples, when it is tested against Iodine-32 encoded exfiltration). We also provide the classifier examples generated by DOLOS (for example, we feed malicious queries generated with DOLOS that is trained with GT or the ISI-rdns benign dataset, but

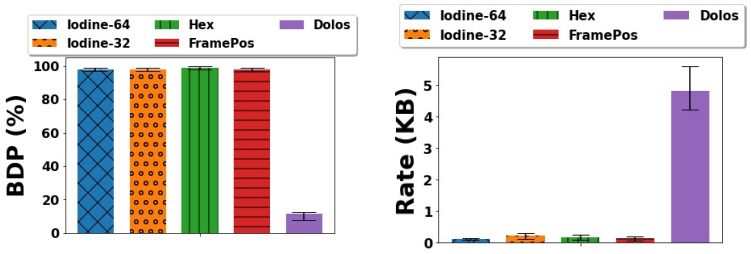

Figure 4.5: On the left is the blackbox detection probability when baseline methods use a constant exfiltration rate commensurate with the average rate of Dolos with deployed anomaly and rule based detection methods. On the right is the maximum rate that baselines can send with a fixed BDP of 0.15.

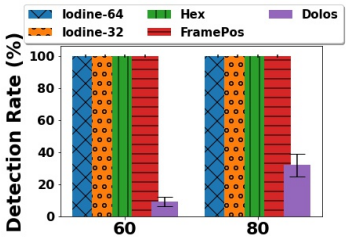

Figure 4.6: Detection rate with Jawad et al., and Ishikura et al., with multiple exfiltration sites (60 and 80 sites).

test them with a different set of queries that are generated with either the same or the other dataset). One can think of this as providing some form of adversarial training [28] to these defenses, which make them very powerful. We hypothesize that since Dolos generates different encodings in each training instance (they all look similar to benign but are different), it is effective even with such whitebox defenses.

Attack and defense setups. We assume the worst outcome upon detection because we consider a blackbox setting, i.e., the defense blocks the primary domain that is flagged. Thus, Dolos monitors and uses the victim host's DNS query patterns to tune the online rates of fake and spurious queries. Some of the defenses we consider seek to detect individual anomalous queries (i.e., ZeekQ and Jawad *et al.*, Chen *et al.*, Buczak *et al.* and Liu *et al.*). We consider queries to the same domain as a flow and incorporate a rule, wherein if the percentage of flagged queries in a flow (denoted as PFQ) exceeds a threshold, the flow (domain) is flagged and the attack is detected. We use the following metrics to evaluate Dolos.

- *Blackbox detection probability (BDP)*: We compute the probability that the exfiltrating primary domain(s) is detected by "at least" one of the defenses. This probability is given by $1 - \prod_i (1 - p_i)$, where p_i is the probability of being detected by defense method d_i.
- *AUC score*: To measure the stealth of DOLOS, we measure a defense's ability to distinguish benign and malicious traffic by using the receiver operating characteristic (ROC) curve, and we report the area under that curve (AUC) score. A low AUC score means that the defense is poor in performing the distinction, which is good for the attacker (e.g., DOLOS).

Default settings. Unless otherwise specified, by default we use DOLOS (ISI-rdns) that is trained on the Text dataset, and defenses that are trained on ISI-host datasets. The results are consistent in behavior with the other datasets and with DOLOS (GT), and we showcase samples of several of those. We also use a single exfiltration domain by default.

4.6.2 Evaluation results

Due to space limits, we present the core results relating to stealth and exfiltration speed of DOLOS in this section.

Holistic evaluation of Dolos. We evaluate DOLOS holistically against baselines as they are used today. We apply all of the rule based and anomaly detection based defenses sequentially (together) and compute the blackbox detection probability (BDP). The defenses are trained on the ISI-host dataset. We exclude the classification methods in computing these plots (i.e., [43, 48, 154]) because they detect the baselines almost surely regardless of the rate they use, based on simply the features of the encoding, while DOLOS is unaffected

(details are discussed later in Table 4.2). Our experiments, upon including these defenses, showed that the absolute performance with Dolos was unchanged from what is discussed below; however, the BDP with the baselines was ≈ 1.0 regardless of the rate, thus, precluding them from exfiltrating almost any data. Our evaluations are with all considered exfiltrated datasets together (i.e., Text, credit cards, logs and Images) and we report the average performance with confidence intervals in Figs. 4.5 and 4.6.

The BDP of baselines are almost close to 1 while* Dolos *experiences a BDP of only 0.12, when they exfiltrate data at the same rate: We perform multiple experiments using Dolos to exfiltrate a set of files from a different exfiltration dataset (e.g., medical images, credit cards) in each run, with an average data size of $\approx 5KB$ from a single host to one external site; this takes on average, 12 hours. We exfiltrate the same file(s) with the baselines, with the *same average rate* of Dolos. To set this rate, we vary the chunk length and choose the longest one that does not flag any of the methods (making query length a non-factor). We then choose a constant rate given this query length, to commensurate with the average rate of Dolos. We see that, at this rate, the BDP of the baselines are almost close to 1 (> 0.95); the baselines have many disadvantages compared to Dolos (e.g., their encoding, fixed rate, lack of spurious queries) and these tend to trigger at least one defense. In contrast, DOLOS has a BDP of just 0.12.

If the baselines use a low rate to avoid detection,* Dolos *can exfiltrate data 25× faster (with an even lower BDP): Now, in contrast to keeping the rate fixed, we fix the tolerable BDP. We search for the most conservative rate that keeps the BDP to below 0.15 for all the methods (similar to what Dolos achieved in the prior experiment).

In such a case, we see in Fig. 4.5 that Dolos is able to transfer 25 times more data than the baselines in a given fixed time.

Increasing the number of exfiltration sites helps* Dolos *boost its exfiltration rate, but does not help baselines. Next, we examine the exfiltration of files from a single host to multiple exfiltration sites in the control of the attacker. We only use Jawad et al., and Ishikura et al., as our defenses since these allow us to explicitly showcase the impact of increasing the number of remote sites. Jawad et al., imposes a maximum length constraint on the queries sent by the exfiltration methods. Ishikura et al., counts the number of cache misses per host, which can increase as the number of external sites to which the attacker sends queries increases. Again, for the baselines, we choose the maximum length that does not flag Jawad et al., ensuring that rate and the number of external sites are the only factors that influence detection. Dolos uses its rate tuning to be consistent with the hosts's query rates. The results are shown in Fig. 4.6. The baselines are almost surely detected because unlike Dolos, they send at fixed rate (increased as a consequence of the length limit due to Jawad at al.) and hence, often exceed the cache miss limit. There is an increase in the detection probability of Dolos as well, because Ishikura et al., counts the cache misses across all its connections to the plurality of sites (in fact, as the number of sites increase the detection rate is likely to increase). However, this increase is modest as seen in Fig. 4.6.

Stealth against individual defenses. We evaluate Dolos 's encoding in evading defenses. We consider exfiltration to a single domain, since most defenses are domain specific. We use the two versions of Dolos (i.e., Dolos (GT) and Dolos (ISI-rdns)) in this exper-

iment. We train the anomaly detectors with the two training datasets exclusively used for defenses (see §4.6.1). Classifiers have access to the attacker's encoding models. To zoom in on the benefits of DOLOS's encoding, *we empower the baselines with* DOLOS*'s rate tuning module (RT)*; in essence they perform markedly better than they would in their native forms (as demonstrated later).

Rule based detection methods have low detection rates on all encoding methods (including baselines) if empowered with* Dolos*'s rate tuning. Zeek does not detect any method because the maximum length of the queries and the rate of unique queries, are consistent with those at the host. Ishikura et al [116] fails to detect exfiltration as long as the exfiltration rate is consistent with benign DNS query generation on the host (cache misses rarely exceed threshold in the time window chosen by the method). ZeekQ does not detect any of the encoding methods including DOLOS, because the percentage of numerical characters conform to its threshold. Paxson *et al.,* has a slightly higher detection probability on methods that aggressively pursue encoding efficiency (the compressed volume is higher). Thus, both Iodine-64 and Iodine-32 which have higher information per bit compared to the others (regardless of query length), are more likely to be detected.

Classification methods almost always detect baselines but* Dolos *evades them because of its ability of generative diverse code books. Buczak et al. [43], Liu et al. [154], Chen et al. [48] learn and thereby almost surely detect signatures generated by the baseline methods (Table 4.2); however, DOLOS goes undetected since it creates diverse sets of codes by simply re-training the encoding-decoding framework. Thus, these defenses are unable to build a signatures of traffic from DOLOS. We emphasize that these classification

methods are whitebox i.e., they have access to the model architecture and the dataset used to train DOLOS; in spite of it, these classification methods fail to detect DOLOS queries, because DOLOS generates different codes with each fresh training. The locations of the very small fraction of queries using Base-32 encoding with DOLOS are staggered depending on the input, and even if a few of these are detected, these defenses cannot easily categorize a domain as malicious.

Anomaly detection methods are effective in detecting baselines but fail to detect* Dolos *due to the similarity of the queries it generates, to benign traffic. Nadler *et al.* [177] detects high entropy codes (e.g., Base-64, and Base-32) and thus is effective in detecting the Iodine variants. The other methods including DOLOS are less easily detectable; specifically, the entropy of DOLOS is very similar to that of benign queries. Jawad *et al.* [10] achieves much higher detection rates with the baselines (> 0.5 detection probability), but DOLOS goes undetected. This is because even when considering multiple features, DOLOS's encoding largely resembles benign traffic enabling it evade even this arguably strongest among defenses that do not need to be trained with malicious samples.

AUC scores. We assess the ability of the encoding methods in generating queries indistinguishable from benign traffic, using AUC scores (see §4.6.1) in Table 4.4. Since, Jawad *et al.,* considers the length of a query in making an inference, to ensure that length is a non-factor in triggering anomalies (only the encoding matters), we impose that malicious traffic of the other baselines are also consistent with the benign query lengths (DOLOS's GAN based encoding ensures this). We use multiple ROC curves where, in each, fake and benign queries of equal length are plotted. *On average* DOLOS *has* $\approx 2\times$ *lower AUC score when*

Table 4.2: Detection probability of defensive methods against the considered attacks. DOLOS evade all detection methods because of the similarity of its encoding to benign traffic; the baselines are flagged by at least a sub-set of the defenses.

Method	Anomaly Detection methods				Classification methods					
	Nadler et al		Jawad et al		Buczak et al		Liu et al		Chen et al	
	Thapar	ISI-HOST	Thapar	ISI-HOST	Thapar	ISI-HOST	Thapar	ISI-HOST	Thapar	ISI-HOST
Iodine-64 + (RT)	0.75	0.59	1	0.98	0.89	0.9	0.92	0.94	0.94	0.96
Iodine-32 + (RT)	0.25	0.32	0.51	0.54	0.88	0.88	0	0	0.94	0.96
Hex + (RT)	0.03	0.02	0.53	0.58	0.92	0.92	0	0	0.94	0.96
FramePos + (RT)	0	0.09	0.57	0.87	0.92	0.92	0.89	0.91	0.94	0.96
DOLOS (GT)	0.04	0.01	0.03	0.06	0	0	0	0	0	0
DOLOS (ISI-rdns)	0.04	0.02	0.02	0.04	0	0	0	0	0	0

Table 4.3: Detection probability of rule based defensive methods against attacks (empowered with DOLOS's rate tuning).

Method	Rule Based Methods							
	Zeek		ZeekQ		Ishikura		Paxson et al	
	Thapar	ISI-HOST	Thapar	ISI-HOST	Thapar	ISI-HOST	Thapar	ISI-HOST
Iodine-64 + (RT)	0	0	0	0	0	0	0.17	0.3
Iodine-32 + (RT)	0	0	0	0	0	0	0.1	0.16
Hex + (RT)	0	0	0	0	0	0	0	0.07
FramePos + (RT)	0	0	0	0	0	0	0	0
DOLOS (GT)	0	0	0	0	0	0	0	0.08
DOLOS (ISI-rdns)	0	0	0	0	0	0	0	0.06

Table 4.4: AUC scores of encoding methods with Jawad et al.

Method	Text	Credit Cards	Logs	Images
Iodine-64	0.996	0.996	0.996	0.996
Iodine-32	0.63	0.62	0.68	0.68
Hex	0.995	0.995	0.995	0.995
FramePos	0.998	0.998	0.998	0.998
DOLOS	0.34	0.35	0.34	0.38

compared against the best baseline (i.e., Iodine-32) when tested against Jawad et al. *[10].* The lower AUC with DOLOS implies that the baseline encoding methods are more likely to be detected than DOLOS even if query length is not a factor, i.e., the encoding with DOLOS can better evade detection compared to baselines.

A case study from previous DNS exfiltration attacks. In a previous attack [109], attackers were able to steal 40M credit cards from US Target stores in 2013 (estimated to be from 1790 stores at that time [265]). The total amount of exfiltrated data was ≈ 4GB.

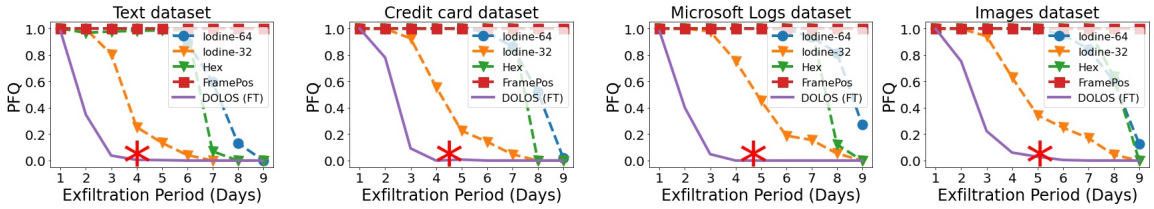

Figure 4.7: Jawad *et al.* defense [10] PFQ on Text, Credit Card, and Logs and Images datasets, respectively, for varying exfiltration rates. The PFQ when using Dolos is much lower than encoding baselines. The red stars represent Dolos's (holistic) performance.

Motivated by this, we ask how fast we can exfiltrate this amount of data compared to the baselines when the attacker compromises the same number of devices. We fix the number of queries per day to 80 and transmit to 120 remote servers, and exfiltrate a 4GB file over varying number of days. We evaluate Dolos (with the same fixed rate (Dolos (FT)) along with the baselines with Jawad *et al.*; this is the best performing defense (other defenses are inferior as seen in Table 4.2) from those that do not need malicious samples for training. We also show how holistic Dolos performs.

Dolos *can exfiltrate data more than 2.5 × faster, than the baselines, without being detected, when evaluated with Jawad* et al. *[10].* We consider the fastest time to exfiltrate a 4GB file with different PFQ thresholds (see § 4.6.1). The results in Fig. 4.7 show that Dolos exfiltrates data much faster and yet is undetected. For example, with the logs dataset, Dolos can potentially go almost undetected even when completing exfiltration in 3 days; in comparison, the closest baseline takes over 8 days if it is to go undetected. In practice, the holistic version of Dolos (shown by the red stars) uses rate tuning (the best fixed rate is unknown) and so performs slightly worse. Importantly, if any of the baselines was to exfiltrate data within 4.5 days (worst case with full Dolos), they would almost surely be detected. We also want to emphasize that this is only with the

defense by Jawad at al. As discussed earlier, if a plurality of defenses are used, the baselines almost always are detected, unlike DOLOS (e.g., with classifiers).

Queries with decoding errors. Next, we report the percentage of queries that were handled by DOLOS's error recovery module. The percentages of such queries for Text, credit cards, Logs and Images datasets are 8%, 5%, 4% and 18%, respectively. While images have the highest error rates (in comparison with others) the rate is still very small to significantly affect stealth.

4.6.3 Dolos Complexity

We had earlier discussed the time for training DOLOS offline (§ 4.6.1). Here, we examine its runtime complexity.

Clustering. The complexity of DOLOS's clustering depends on the benign query rate of the victim and the number of unique domains. Computations are amortized over a time window and require low CPU workloads. The average number of clusters per domain is 70 against each of which, each query is compared; this takes 1.5ms. With regards to *space overhead*, we store the number of clusters for each unique domain, as well as their representative queries. The maximum needed space for a 24-hour window is 0.6MB (600KB), which is insignificant with respect to today's computers.

Data encoding. We measure the average encoding duration at test time, on a 2.9 GHz laptop CPU with 3 different batch sizes (1, 8 and 16). The average encoding times are 1.4s, 0.452s and 0.28s, respectively. Since the exfiltration queries are sent at time scales of seconds/minutes to avoid detection, this overhead has a negligible influence on the rate. With regards to the *space overhead*, the model sizes for text, credit card and logs models

and images are 59M, 22M and 18M, 30M, respectively. On average, the size of the encoder is 35MB, which is very small given the disk and memory of today's machines. The size of the decoder is the same as the encoder associated with a given dataset. The average decoding times for batch sizes (1,8 and 16) are 1.5s, 0.49s and 0.31s, respectively. We break up the model into multiple files to reduce the storage footprint; the model can be composed from these scattered files in the memory during operation with negligible additional time. Further optimization to smaller and cheaper architectures (e.g., using model pruning or knowledge distillation) is viable but left for future work.

4.6.4 Real implementation details and results

We host a domain we purchased from godaddy.com, on two name servers as described in RFC [174]. The two name servers are Windows server 2019 virtual machines deployed on Microsoft Azure. We use the default Microsoft DNS server [170], and we generate *CNAME* DNS records for the generated domain as **.dolos.com* (fake name for anonymity reasons). This makes the server respond to DNS queries with any subdomain under *dolos.com* with the default configured IP address. We use the default DNS logging system to decode queries to their original representations. Client queries (regardless of whether they are benign or malicious) are generated using a Mac pro laptop with `nslookup`. We use three popular public DNS resolvers: Google (`IP:8.8.8.8`), Control D (`IP:76.76.10.0`) and CloudFlare (`IP:1.1.1.1`). In addition we also experiment with the DNS resolver (owned by our institution). We capture DNS queries on the local machine and extract the features required to make an inference using the defenses we considered in § 4.6.1. In our experiments we exfiltrate a 10KB text file with an average chunk length of 90 characters. We

repeat the exfiltration with DOLOS and each baseline, but the benign queries are only those that are naturally generated. These constraints are imposed because we do not want to cause unintentional damage to either the public or our institutional DNS resolvers (e.g., DDoS attacks).

Real Implementation results. Since we are unaware if these real systems deploy defenses, we also pass the traffic through Jawad et al., for safety. Flagged queries are stopped and do not reach the destination. We use all considered name servers in this experiment. Finally, we ended up retrieving all fake queries that were not detected by Jawad et al., which suggests that the public DNS resolvers do not employ any potent defense (although we cannot verify this explicitly). We will investigate this in the future by talking to administrators or other means (e.g., security blogs).

4.7 Discussion

Defending against `BigEye`. One potential defense against `BigEye` is to use program analysis to identify programs responsible for creating DNS queries for exfiltration. One can check if a program accesses (i) confidential files and (ii) inputs those to a processing engine. However, running such analyses on all programs on all hosts is expensive. Instead, one can monitor the DNS queries and use a detector or ensemble of detectors with somewhat lower detection thresholds (i.e., higher detection rate of attacks but high false positives). Expensive program analysis can be used as a second filtration layer on only those programs that generated these queries.

Exfiltrating multiple types of data. To exfiltrate multiple types of data, one can train and use multiple encoders. At run time, the malware can include a codeword to indicate the type of data being exfiltrated. The remote server then uses the appropriate decoder for recovery.

Whitelists and their impact on `BigEye`. Enterprises could use whitelists to block access to `BigEye`'s remote server. Today, the deployed whitelists specify the *only* domains that users can access (i.e., all other domains are inaccessible/blocked by default). [179] suggests that the top (most popular) one million domains could be used as such, in a whitelist. However, we argue that such whitelists are highly impractical in the real world, because access patterns to domains/websites could vary significantly over time. This makes it infeasible for network operators to keep such lists up to date without inducing blockages to benign domains. Google reports that about 4% of the 2 billion domains are accessed with between 10 and 1000 visits, in a month [228]. This is about 80 million websites and not all of these can be in the top-1-million whitelist described in [179]. In other words, having whitelists with a million pages causes more problems for defenses in terms of false positives and thus is not useful against `BigEye`.

Out-of-order packet delivery. In-order packet delivery from the compromised host to the remote domain is not guaranteed by the DNS protocol. To ensure delivery guarantees, new attributes such as sequence numbers may be needed. This aspect is left for future work. We highlight that `BigEye` decodes each query independently. Thus, we believe that using the decoded queries, an ML model can assess the received queries to determine whether they were received in order and rectify the order if necessary.

4.8 Related Work

Text generation. GPT [205] and BERT [64] are ML methods to synthesize text similar to training text. However, our problem differs since we seek a latent space representation, largely decodable to its raw form. Similarly, GANs have been used for text generation [139, 278] but do not handle multiple goals (mimicking benign traffic and ensuring decodability). Recent work uses multiple generators and discriminators with different loss functions in order to achieve a single objective (e.g., generating better synthetic images [86, 231]). However, DOLOS seeks to fulfil multiple objectives.

Adversarial Perturbations. There is work on perturbing inputs to deceive a neural network (e.g., [244, 283]). Such methods cannot be readily applied to modify exfiltration data. NIDSGAN [283] perturbs packet headers while adhering to domain constraints to deceive ML-based intrusion detection systems. However, they only have one objective (evasion).

Encoder-decoder frameworks. ML-based encoder-decoder frameworks have been studied. For example, [76] builds such a framework to send text data over an erasure channel. This work differs from ours in two ways. First, the latent representations are continuous whereas in our case, they are discrete (they form the fake queries). Beyond this, our approach has constraints on the latent space (it should resemble benign traffic). A work close to ours is [124], where the authors design an end-to-end communication system over an erasure channel but constrain the latent representation to binary values. The problem differs from ours because, in addition to constraining the representation to characters used by DNS, we also have additional requirements (e.g., stealth and decodability).

Detection. As discussed, defenses against DNS exfiltration are either rate based [71, 128]

or encoding based [61, 123, 140, 149, 163, 276]. These methods were either considered in our evaluations in § 4.6.2 or are very similar to those considered and we expect them to fail in detecting DOLOS.

DNS Exfiltration tools. Today's DNS exfiltration tools such as DNSmessanger [42], DNSteal [83] and Iodine [54], use DNS query types that do not conform with benign traffic and/or use the traditional encoding (e.g., Base-64, Base-32 and Hex) that are inefficient and detectable. For example, DNSmessager uses DNS type `TXT`, which, as discussed, is easily detectable.

4.9 Conclusions

In this work, we show that contrary to the common belief that current defenses have succeeded in curbing DNS exfiltration, our GAN-guided approach, DOLOS, can achieve successful exfiltration. DOLOS encodes the exfiltration data such that it not only mimics benign traffic to fool defenses but also achieves significantly faster exfiltration than what is possible today. In addition, it includes a rate tuning module that is key in preventing the attack from being detected in a blackbox setup. Finally, DOLOS also ensures that exfiltrated data is fully decodable at an external site like with traditional methods. Our evaluations show that DOLOS can exfiltrate data 25 $\times$ faster than the baselines, if the detection probability needs to be kept low (< 0.15).

Algorithm 2 Training DOLOS Encoding

Input: exfiltration and benign DNS datasets , acc_D

Input: Validation dataset (V_D), Validation Model (V_M)

Input: Validation model fooling rate threshold (β)

Input: Training time out threshold (H)

for γ in range (0.5,1.5,0.1) **do**

Initialize Dec, Dis, Enc with latent space of size $\gamma * L^{m_i}$

while True **do**

(1) Sample batches from exfiltration dataset and benign dataset of size B.

(2) Update the `Discriminator` as follows: $\nabla_{\theta_{Dis}} \frac{1}{B} \sum_k^B [log(Dis(x^k)) + log(1 - Dis(Enc(z^k)))$.

(3) Update the `Encoder`: $\nabla_{\theta_{Enc}} \alpha \frac{1}{B} \sum_k^B [log(1 - Dis(Enc(z^k))) + (1 - \alpha) * \frac{1}{L^{m_i}} \sum_j log(p_{\hat{c^j}})$.

(4) Update `Decoder` with $\nabla_{\theta_{Dec}} \frac{1}{L^{m_i}} \sum_j log(p_{\hat{c^j}})$.

if $\frac{1}{|V_D|} \sum_{v \in V_D} \mathbf{1}($ $argmax \quad Dec(Enc(v)) = v$ $) \geq acc_D$ & $V_M(Enc(V_D)) \geq \beta$ **then**

return trained DOLOS

end if

if # of batches $\geq H$ **then**

break {Need a bigger encoding size}

end if

end while

end for=0

Algorithm 3 DOLOS clustering methodology

```
Input : list requests to a domain (Queries), threshold
Initialize clusters set (ClusSet)
for newQ in Queries do
    for q in ClusSet do
        if JaccardSimilarity(newQ,q) ≥ threshold then
            Break {Found a match}
        end if
    end for
    if no match then
        ClusSet ← newQ {Create new cluster with query newQ}
    end if
end for=0
```

Chapter 5

Conclusions

In this dissertation, we identified multiple networking applications in constrained environments where machine learning can advance the state-of-the-art beyond traditional methods. We first presented `BigEye`, which detects and summarizes key global events from distributed data sources using a very limited amount of data. We analytically showed that `BigEye` can achieve the same accuracy as cases where data is available centrally, with less than 1% of the entire data volume empirically demonstrated.

Next, we introduced `AcTrak`, a reinforcement learning-based control system for steering camera movements within a specified area of interest. The trained RL agent orchestrates camera movements to effectively balance the trade-off between zoom levels.

Finally, we introduced Dolos, demonstrating how ML-based malware can evade DNS exfiltration defenses to exfiltrate data undetected. Dolos is a framework that trains an ML-based encoder and decoder to encode exfiltrated data to resemble benign traffic, thereby evading defenses and ensuring the encoded data can be decoded back to its original

representation. Additionally, Dolos uses heuristic approaches to dynamically adjust the exfiltration rate at runtime to evade detection.

Bibliography

[1] Loghub: A collection of system log datasets for intelligent log analysis. https://github.com/logpai/loghub, 2019.

[2] Kaggle. https://www.kaggle.com/datasets/manann/quotes-500k?select=quotes.csv, 2020.

[3] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[4] Evangelos Papalexakis Lance Kaplan Srikanth V. Krishnamurthy Tarek Abdelzaher Abdulrahman Fahim, Ajaya Neupane. Edge-assisted detection and summarization of key global events from distributed crowd-sensed data. In *Proceedings of IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019.

[5] Lawrence Abrams. New royal ransomware emerges in multi-million dollar attacks. https://www.bleepingcomputer.com/news/security/new-royal-ransomware-emerges-in-multi-million-dollar-attacks/, 2022.

[6] National Security Agency. Adopting encrypted dns in enterprise environments. https://media.defense.gov/2021/Jan/14/2002564889/-1/-1/0/CSI_ADOPTING_ENCRYPTED_DNS_U_OO_102904_21.PDF, 2021.

[7] Charu C Aggarwal and Karthik Subbian. Event detection in social streams. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 624–635. SIAM, 2012.

[8] Charu C Aggarwal and Philip S Yu. A framework for clustering massive text and categorical data streams. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 479–483. SIAM, 2006.

[9] Amir M. Ahmadian and Musard Balliu. Dynamic policies revisited. In *IEEE European Symposium on Security and Privacy*, 2022.

[10] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. Monitoring enterprise dns queries for detecting data exfiltration from

internal hosts. *IEEE Transactions on Network and Service Management*, 17(1):265–279, 2019.

[11] Jeff Ahrenholz, Claudiu Danilov, Thomas R Henderson, and Jae H Kim. Core: A real-time network emulator. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE, 2008.

[12] Akamai. Power and protect life online. urlhttps://www.akamai.com/, 2000.

[13] Akamai. Dns: The easiest way to exfiltrate data? `https://www.akamai.com/blog/security/dns-the-easiest-way-to-exfiltrate-data`, 2022.

[14] Isabela Albuquerque, Joao Monteiro, Thang Doan, Breandan Considine, Tiago Falk, and Ioannis Mitliagkas. Multi-objective training of generative adversarial networks with multiple discriminators. In *International Conference on Machine Learning*, pages 202–211. PMLR, 2019.

[15] Fatemah Alharbi, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael Abu-Ghazaleh. Dns poisoning of operating system caches: Attacks and mitigations. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2851–2863, 2022.

[16] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Zia Ush Shamszaman, Thu-Le Pham, Feng Gao, Keith Griffin, and Alessandra Mileo. Real-time data analytics and event detection for iot-enabled communication systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 42:19–37, 2017.

[17] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *ACM SIGIR Forum*, volume 51, pages 185–193. ACM, 2017.

[18] alsecuritycamera. axis outdoor multi sensor ip security camera. `{https://www.a1securitycameras.com}`.

[19] Amazon. AWS for the Edge: Bringing data processing and analysis closer to endpoints. `{https://aws.amazon.com/edge/}`. [Online; accessed June-1-2021].

[20] Amazon.com. Amazon.com. `{https://www.amazon.com/}`.

[21] Apollo. Moscow says Twitter ready to store data of users on Russian servers despite concerns over surveillance . `{http://apollo2.cs.illinois.edu/index.html}`, Nov, 2014. [Online; accessed Oct-11-2018].

[22] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[23] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

[24] Shayan Modiri Assari, Haroon Idrees, and Mubarak Shah. Human re-identification in crowd videos using personal, social and environmental constraints. In *European Conference on Computer Vision*. Springer, 2016.

[25] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[26] Avipas. Avipas model av-1081 manual. {https://e7aba150-670b-4b8b-9a25-311a84251d5f.filesusr.com/ugd/6b6a18_34ff6afc20914be9943327dc3f5a6211.pdf}.

[27] Andrew D Bagdanov, Alberto Del Bimbo, Walter Nunziati, and Federico Pernici. A reinforcement learning approach to active camera foveation. In *ACM international workshop on video surveillance and sensor networks*, 2006.

[28] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness. *arXiv preprint arXiv:2102.01356*, 2021.

[29] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[30] James Benhardus and Jugal Kalita. Streaming trend detection in twitter. *International Journal of Web Based Communities*, 9(1):122–139, 2013.

[31] Michael S Bernstein, Bongwon Suh, Lichan Hong, Jilin Chen, Sanjay Kairam, and Ed H Chi. Eddi: interactive topic-based browsing of social status streams. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 303–312. ACM, 2010.

[32] Amlaan Bhoi. Monocular depth estimation: A survey. *arXiv preprint arXiv:1901.09402*, 2019.

[33] bhphotovideo.com. bhphotovideo.com. {https://www.bhphotovideo.com/}. [Online; accessed June-29-2021].

[34] Jingwen Bian, Yang Yang, and Tat-Seng Chua. Multimedia summarization for trending topics in microblogs. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1807–1812. ACM, 2013.

[35] Andrea Bianco, Paolo Giaccone, Reza Mashayekhi, Mario Ullio, and Vinicio Vercellone. Scalability of onos reactive forwarding applications in isp networks. *Computer Communications*, 102:130–138, 2017.

[36] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[37] Niccolò Bisagno, Alberto Xamin, Francesco De Natale, Nicola Conci, and Bernhard Rinner. Dynamic camera reconfiguration with reinforcement learning and stochastic methods for crowd surveillance. *Sensors*, 20(17):4691, 2020.

[38] bleepingcomputer.com. Iranian hackers exposed in a highly targeted espionage campaign. `https://www.bleepingcomputer.com/news/security/iranian-hackers-exposed-in-a-highly-targeted-espionage-campaign/`, 2022.

[39] Alexander Boettcher and Dongman Lee. Eventradar: A real-time local event detection scheme using twitter stream. In *2012 IEEE International Conference on Green Computing and Communications*, pages 358–367. IEEE, 2012.

[40] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

[41] Seth Bromberger. Dns as a covert channel within protected networks. *National Electronic Sector Cyber Security Organization (NESCO)(Jan., 2011)*, 2011.

[42] Edmund Brumaghin. Covert Channels and Poor Decisions: The Tale of DNS-Messenger. `http://blog.talosintelligence.com/2017/03/dnsmessenger.html`, 2017. [Online; accessed March-15-2022].

[43] Anna L Buczak, Paul A Hanke, George J Cancro, Michael K Toma, Lanier A Watkins, and Jeffrey S Chavis. Detection of tunnels in pcap data by random forests. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pages 1–4, 2016.

[44] Luke Burns. FAQ: The "snake fight" portion of your thesis defense. *McSweeney's*, Nov 2010.

[45] Joseph Carson. Privilege escalation on linux: When it's good and when it's a disaster (with examples). `https://delinea.com/blog/linux-privilege-escalation`, 2020.

[46] Surajit Chaudhuri and Raghav Kaushik. Extending autocompletion to tolerate errors. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 707–718, 2009.

[47] Chung-Hao Chen, Yi Yao, David Page, Besma Abidi, Andreas Koschan, and Mongi Abidi. Camera handoff and placement for automated tracking systems with multiple omnidirectional cameras. *Computer Vision and Image Understanding*, 114(2):179–197, 2010.

[48] Shaojie Chen, Bo Lang, Hongyu Liu, Duokun Li, and Chuan Gao. Dns covert channel detection method using the lstm model. *Computers & Security*, 104:102095, 2021.

[49] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 759–768. ACM, 2010.

[50] Flavio Chierichetti, Jon M Kleinberg, Ravi Kumar, Mohammad Mahdian, and Sandeep Pandey. Event detection via communication pattern analysis. In *Proceedings of ICWSM*, 2014.

[51] Jinyoung Choi and Han Bohyung. Mcl-gan: Generative adversarial networks with multiple specialized discriminators. 2019.

[52] CloudFlare. Http policies. `https://developers.cloudflare.com/cloudflare-one/policies/filtering/http-policies/`, 2020.

[53] Cloudflare. Cloudflare Resource Hub. `https://www.cloudflare.com/resource-hub/?resourcetype=Whitepaper`, 2022. [Online; accessed Dec-1-2022].

[54] code.kryo.se. Iodine:code.kryo.se. `https://github.com/yarrick/iodine`, 2021.

[55] Cash J Costello, Christopher P Diehl, Amit Banerjee, and Hesky Fisher. Scheduling an active camera to observe people. In *Proceedings of the ACM 2nd international workshop on Video surveillance & sensor networks*, pages 39–45, 2004.

[56] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006.

[57] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.

[58] Dahuq. Dahuq fish eye camera.

[59] Jason Haddix Daniel Miessler and g0tmi1k. Seclists. `https://github.com/danielmiessler/SecLists`, 2022.

[60] darknet. YOLO: Real-Time Object Detection. `{https://pjreddie.com/darknet/yolo/}`. [Online; accessed August-10-2022].

[61] Anirban Das, Min-Yi Shen, Madhu Shashanka, and Jisheng Wang. Detection of exfiltration and tunneling over dns. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 737–742. IEEE, 2017.

[62] Greire Payen de La Garanderie, Amir Atapour Abarghouei, and Toby P Breckon. Eliminating the blind spot: Adapting 3d object detection and monocular depth estimation to 360 panoramic imagery. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 789–807, 2018.

[63] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[64] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[65] Chong Ding, Jawadul H Bappy, Jay A Farrell, and Amit K Roy-Chowdhury. Opportunistic image acquisition of individual and group activities in a distributed camera network. *IEEE transactions on circuits and systems for video technology*, 27(3):664–672, 2016.

[66] Yann Disser, Max Klimm, Nicole Megow, and Sebastian Stiller. Packing a knapsack of unknown capacity. *SIAM Journal on Discrete Mathematics*, 31(3):1477–1497, 2017.

[67] B. Doroodgar and G. Nejat. A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments. In *2010 IEEE International Conference on Automation Science and Engineering*, pages 948–953, 2010.

[68] Harris Drucker. Improving regressors using boosting techniques. In *ICML*.

[69] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1285–1298, 2017.

[70] EfficientIp. IDC 2021 Global DNS Threat Report. `https://www.efficientip.com/resources/idc-dns-threat-report-2021/`, 2021. [Online; accessed April-20-2022].

[71] Wendy Ellens, Piotr Żuraniewski, Anna Sperotto, Harm Schotanus, Michel Mandjes, and Erik Meeuwissen. Flow-based detection of dns tunnels. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 124–135. Springer, 2013.

[72] EnterpriseDT. How to secure your sftp server. `https://enterprisedt.com/blogs/completeftp/how-to-secure-sftp-server/`, 2020.

[73] Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi. Automatic spelling correction for resource-scarce languages using deep learning. In *Proceedings of ACL 2018, Student Research Workshop*, pages 146–152, 2018.

[74] Abdelrahman Fahim, Ajaya Neupane, Evangelos Papalexakis, Lance Kaplan, Srikanth V Krishnamurthy, and Tarek Abdelzaher. Edge-assisted detection and summarization of key global events from distributed crowd-sensed data. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 76–85. IEEE, 2019.

[75] Vivian Fang, Lloyd Brown, William Lin, Wenting Zheng, Aurojit Panda, and Raluca Ada Popa. CostCO: An automatic cost modeling framework for secure multi-party computation. In *IEEE European Symposium on Security and Privacy*, 2022.

[76] Nariman Farsad, Milind Rao, and Andrea Goldsmith. Deep learning for joint source-channel coding of text. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2326–2330. IEEE, 2018.

[77] Mehdi Fatemi, Mary Wu, Jeremy Petch, Walter Nelson, Stuart J Connolly, Alexander Benz, Anthony Carnicelli, and Marzyeh Ghassemi. Semi-markov offline reinforcement learning for healthcare. In *Conference on Health, Inference, and Learning*, pages 119–137. PMLR, 2022.

[78] Aaron Faulkenberry, Athanasios Avgetidis, Zane Ma, Omar Alrawi, Charles Lever, Panagiotis Kintis, Fabian Monrose, Angelos D Keromytis, and Manos Antonakakis. View from above: Exploring the malware ecosystem from the upper dns hierarchy. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 240–250, 2022.

[79] Wei Feng, Chao Zhang, Wei Zhang, Jiawei Han, Jianyong Wang, Charu Aggarwal, and Jianbin Huang. Streamcube: hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1561–1572. IEEE, 2015.

[80] FireEye. Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor. `https://www.fireeye.com/blog/threat-research/`, 2021. [Online; accessed June-1-2021].

[81] Dennis Fisher. Ransomware actors leaning on dns tunneling. `https://duo.com/decipher/ransomware-actors-leaning-on-dns-tunneling`, 2022.

[82] Romain Fouchereau. IDC 2021 Global DNS Threat Report. `https://www.efficientip.com/resources/idc-dns-threat-report-2021/`, 2021. [Online; accessed April-20-2022].

[83] g0dmode. Dnssteal. `https://github.com/m57/dnsteal`, 2015.

[84] The Gaurdian. SolarWinds hack was work of 'at least 1,000 engineers', tech executives tell Senate. `https://www.theguardian.com/technology/2021/feb/23/solarwinds-hack-senate-hearing-microsoft`, 2021 @articlegoodfellow2014generative, title=Generative adversarial nets, author=Goodfellow, Ian and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Yoshua, journal=Advances in neural information processing systems, volume=27, year=2014 . [Online; accessed September-20-2021].

[85] Gdatasoftware. New FrameworkPOS variant exfiltrates data via DNS requests. `https://www.gdatasoftware.com/blog/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests`, 2014. [Online; accessed Feb-26-2022].

[86] Arnab Ghosh, Viveka Kulharia, Vinay P Namboodiri, Philip HS Torr, and Puneet K Dokania. Multi-agent diverse generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8513–8521, 2018.

[87] Prasanna Giridhar, Shiguang Wang, Tarek F Abdelzaher, Jemin George, Lance Kaplan, and Raghu Ganti. Joint localization of events and sources in social networks. In *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*, pages 179–188. IEEE, 2015.

[88] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[89] Google. Google place autocomplete api. `https://developers.google.com/maps/documentation/places/web-service/autocomplete#maps_http_places_autocomplete_amoeba-py`, 2020.

[90] Google. Google places api. `https://developers.google.com/maps/documentation/places/web-service/overview`, 2020.

[91] Google. kctf vrp setup. `https://google.github.io/kctf/vrp.html`, 2022.

[92] Google. Public kctf responses. `https://docs.google.com/spreadsheets/d/e/2PACX-1vS1REdTA29OJftst8xN5B5x8iIUcxuK6bXdzF8G1UXCmRtoNsoQ9MbebdRdFnj6qZ0Yd7LwQfvYC/pubhtml`, 2022.

[93] grammarly. Great Writing, Simplified]. `https://www.grammarly.com/`, 2020. [Online; accessed July-10-2022].

[94] Chao Gui and Prasant Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 129–143. ACM, 2004.

[95] Adrien Guille and Cécile Favre. Mention-anomaly-based event detection and tracking in twitter. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 375–382. IEEE, 2014.

[96] Sonali P Gulve, Suchitra A Khoje, and Prajakta Pardeshi. Implementation of iot-based smart video surveillance system. In *Computational intelligence in data mining*, pages 771–780. Springer, 2017.

[97] Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y Yen, Runhe Huang, and Xingshe Zhou. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys (CSUR)*, 48(1):7, 2015.

[98] Himanshu Gupta, Vishnu Navda, Samir Das, and Vishal Chowdhary. Efficient gathering of correlated data in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):4, 2008.

[99] Adnan Gutub and Manal Fattani. A novel arabic text steganography method using letter points and extensions. 2007.

[100] Said Hanafi and Arnaud Freville. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2-3):659–675, 1998.

[101] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web (extended abstract). In *Third International Workshop on the Web and Databases (WebDB 2000)*, 2000.

[102] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448*, 2020.

[103] J Dinal Herath, Ping Yang, and Guanhua Yan. Real-time evasion attacks against deep learning-based anomaly detection from distributed system logs. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy*, pages 29–40, 2021.

[104] Here.com. *Location data processing*, (accessed June, 2018).

[105] Dominik Herrmann, Christian Banse, and Hannes Federrath. Behavior-based tracking: Exploiting characteristic patterns in dns traffic. *Computers & Security*, 39:17–33, 2013.

[106] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[107] Hhzzk. Dns-tunnels. `https://github.com/hhzzk/dns-tunnels`, 2017.

[108] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[109] Elise Hu. Hackers stole 40 million credit, debit card numbers from target. `https://www.npr.org/2013/12/19/255559793/hackers-stole-40-million-credit-debit-card-numbers-from-target`, 2013.

[110] Yuheng Hu, Ajita John, Dorée Duncan Seligmann, and Fei Wang. What were the tweets about? topical associations between public events and twitter feeds. In *Proceedings of ICWSM*, 2012.

[111] Ting Hua, Feng Chen, Liang Zhao, Chang-Tien Lu, and Naren Ramakrishnan. Automatic targeted-domain spatiotemporal event detection in twitter. *GeoInformatica*, 20(4):765–795, 2016.

[112] iagox86. Dnscat2. https://github.com/iagox86/dnscat2, 2022.

[113] illinois urbana champaign. *Social sensing tool*, (accessed June, 2018).

[114] Infoblox. Data exfiltration and dns. https://www.infoblox.com/wp-content/uploads/infoblox-whitepaper-data-exfiltration-and-dns-closing-the-back-door.pdf, 2020.

[115] Infoblox. Preventing dns-based data exfiltration. https://www.infoblox.com/wp-content/uploads/infoblox-solution-note-preventing-dns-based-data-exfiltration.pdf, 2020.

[116] Naotake Ishikura, Daishi Kondo, Vassilis Vassiliades, Iordan Iordanov, and Hideki Tode. Dns tunneling detection by cache-property-aware features. *IEEE Transactions on Network and Service Management*, 18(2):1203–1217, 2021.

[117] GI Ivchenko and SA Honov. On the jaccard similarity test. *Journal of Mathematical Sciences*, 88(6):789–794, 1998.

[118] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[119] Mark Jager, Christian Knoll, and Fred A Hamprecht. Weakly supervised learning of a classifier for unusual event detection. *IEEE Transactions on Image Processing*, 17(9):1700–1708, 2008.

[120] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. Panoptes: Servicing multiple applications simultaneously using steerable cameras. In *IPSN*, pages 119–130, 2017.

[121] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[122] Paul Jasek and Bernard Abayowa. Visual sensor network reconfiguration with deep reinforcement learning. *arXiv preprint arXiv:1808.04287*, 2018.

[123] Iram Jawad, Jawad Ahmed, Imran Razzak, and Robin Doss. Identifying dns exfiltration based on lexical attributes of query name. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2021.

[124] Yihan Jiang, Hyeji Kim, Himanshu Asnani, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels. *Advances in neural information processing systems*, 32:2758–2768, 2019.

[125] Chris Johnston. Home Depot: 56 million credit cards compromised. https://www.theguardian.com/business/2014/sep/19/home-depot-56m-credit-card-numbers-compromised, 2014. [Online; accessed June-15-2022].

[126] Kaggle. Covid-19 Image Dataset. {https://www.kaggle.com/datasets/pranavraikokte/covid19-image-dataset}, 2021. [Online; accessed Jan-10-2023].

[127] Kaggle. Disaster tweets dataset. {https://www.kaggle.com/vstepanenko/disaster-tweets}, Jan, 2021. [Online; accessed Jan-05-2021].

[128] A Mert Kara, Hamad Binsalleeh, Mohammad Mannan, Amr Youssef, and Mourad Debbabi. Detection of malicious payload distribution channels in dns. In *2014 IEEE International Conference on Communications (ICC)*, pages 853–858. IEEE, 2014.

[129] Alla Katsnelson. Colour me better: fixing figures for colour blindness. *Nature*, Oct 2021.

[130] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[131] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Introduction to np-completeness of knapsack problems. In *Knapsack problems*, pages 483–493. Springer, 2004.

[132] Karim Khalil, Azeem Aqil, Srikanth V Krishnamurthy, Tarek Abdelzaher, and Lance Kaplan. Nest: Efficient transport of data summaries over named data networks. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2018.

[133] Umair Ali Khan and Bernhard Rinner. Online learning of timeout policies for dynamic power management. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4):1–25, 2014.

[134] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.

[135] Dongchil Kim, Kyoungman Kim, and Sungjoo Park. Automatic ptz camera control based on deep-q network in video surveillance system. In *ICEIC*. IEEE, 2019.

[136] Data Center Knowledge. Twitter Adding More Data Center Space (Again). {https://www.datacenterknowledge.com/archives/2011/09/19/twitter-adding-more-data-center-space-again}, Sept 19, 2011. [Online; accessed Oct-11-2018].

[137] krebsonsecurity.com. Deconstructing the 2014 sally-beauty breach. https://krebsonsecurity.com/2015/05/deconstructing-the-2014-sally-beauty-breach/, 2015.

[138] R Bala Krishnan, Prasanth Kumar Thandra, and M Sai Baba. An overview of text steganography. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6. IEEE, 2017.

[139] Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

[140] Danielle Lambion, Michael Josten, Femi Olumofin, and Martine De Cock. Malicious dns tunneling detection in real-traffic dns data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 5736–5738. IEEE, 2020.

[141] Shuyue Lan, Rameswar Panda, Qi Zhu, and Amit K Roy-Chowdhury. Ffnet: Video fast-forwarding via reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6771–6780, 2018.

[142] Jey Han Lau, Nigel Collier, and Timothy Baldwin. On-line trend analysis with topic models:\# twitter trends detection topic model online. *Proceedings of COLING 2012*, pages 1519–1534, 2012.

[143] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer graphics forum*. Wiley Online Library, 2007.

[144] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Finding Similar Items*, page 68–122. Cambridge University Press, 2 edition, 2014.

[145] Chenliang Li, Aixin Sun, and Anwitaman Datta. Twevent: segment-based event detection from tweets. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 155–164. ACM, 2012.

[146] Rui Li, Kin Hou Lei, Ravi Khadiwala, and Kevin Chen-Chuan Chang. Tedas: A twitter-based event detection and analysis system. In *Data engineering (icde), 2012 ieee 28th international conference on*, pages 1273–1276. IEEE, 2012.

[147] Wei Li, Fan Zhou, Kaushik Roy Chowdhury, and Waleed Meleis. Qtcp: Adaptive congestion control with reinforcement learning. *IEEE Transactions on Network Science and Engineering*, 6(3):445–458, 2018.

[148] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A survey on truth discovery. *ACM Sigkdd Explorations Newsletter*, 17(2):1–16, 2016.

[149] Jianbing Liang, Suxia Wang, Shuang Zhao, and Shuhui Chen. Fecc: Dns tunnel detection model based on cnn and clustering. *Computers & Security*, 128:103132, 2023.

[150] Chih-Yu Lin, Wen-Chih Peng, and Yu-Chee Tseng. Efficient in-network moving object tracking in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 5(8):1044–1056, 2006.

[151] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[152] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[153] Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos "in the wild". In *CVPR 2009*, pages 1996–2003. IEEE, 2009.

[154] Jingkun Liu, Shuhao Li, Yongzheng Zhang, Jun Xiao, Peng Chang, and Chengwei Peng. Detecting dns tunnel through binary-classification based on behavior features. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 339–346. IEEE, 2017.

[155] Sheng Liu, Michael K Reiter, and Vyas Sekar. Flow reconnaissance via timing attacks on sdn switches. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 196–206. IEEE, 2017.

[156] Siyuan Liu, Yunhuai Liu, Lionel Ni, Minglu Li, and Jianping Fan. Detecting crowdedness spot in city transportation. *IEEE Transactions on Vehicular Technology*, 62(4):1527–1539, 2012.

[157] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[158] Chaoyi Lu, Baojun Liu, Zhou Li, Shuang Hao, Haixin Duan, Mingming Zhang, Chunying Leng, Ying Liu, Zaifeng Zhang, and Jianping Wu. An end-to-end, large-scale measurement of dns-over-encryption: How far have we come? In *Proceedings of the Internet Measurement Conference*, pages 22–35, 2019.

[159] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer.

[160] Hans Peter Luhn. Computer for verifying numbers. *US Patent*, 2(950):048, 1960.

[161] Huadong Ma, Dong Zhao, and Peiyan Yuan. Opportunities in mobile crowd sensing. *IEEE Communications Magazine*, 52(8):29–35, 2014.

[162] Yajie Ma, Yike Guo, Xiangchuan Tian, and Moustafa Ghanem. Distributed clustering-based aggregation algorithm for spatial correlated sensor networks. *IEEE Sensors Journal*, 11(3):641–648, 2010.

[163] Samaneh Mahdavifar, Amgad Hanafy Salem, Princy Victor, Amir H Razavi, Miguel Garzon, Natasha Hellberg, and Arash Habibi Lashkari. Lightweight hybrid detection of data exfiltration using dns based on machine learning. In *2021 the 11th International Conference on Communication and Network Security*, pages 80–86, 2021.

[164] VASILENA MARKOVA. What is nxdomain? `https://www.cloudns.net/blog/what-is-nxdomain/#:~:text=NXDOMAIN%20stands%20for%20a%20non,the%20domain%20does%20not%20exist.`, 2024.

[165] Julian John McAuley and Jure Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908, 2013.

[166] Robert McCraith, Lukas Neumann, and Andrea Vedaldi. Calibrating self-supervised monocular depth estimation. *arXiv preprint arXiv:2009.07714*, 2020.

[167] Richard McCreadie, Craig Macdonald, Iadh Ounis, Miles Osborne, and Sasa Petrovic. Scalable distributed event detection for twitter. In *Big Data, 2013 IEEE International Conference on*, pages 543–549. IEEE, 2013.

[168] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[169] Microsoft. Implementing a zero trust security model at microsoft. `https://www.microsoft.com/en-us/insidetrack/implementing-a-zero-trust-security-model-at-microsoft`, 2022.

[170] Microsoft. Quickstart: Installing and configure dns server. `https://learn.microsoft.com/en-us/windows-server/networking/dns/quickstart-install-configure-dns-server?tabs=powershell`, 2022.

[171] Microsoft. White papers on the cloud and Azure. `https://azure.microsoft.com/en-us/resources/whitepapers/search/?type=WhitePaperResource&Page=1`, 2022. [Online; accessed Dec-1-2022].

[172] David Mills et al. Network time protocol. Technical report, RFC 958, M/A-COM Linkabit, 1985.

[173] Mininet. *Mininet.org*, (accessed June, 2018).

[174] P. Mockapetris. *RFC 1035 Domain Names - Implementation and Specification*. Internet Engineering Task Force, November 1987.

[175] Sharon Moltchanov, Ilan Levy, Yael Etzion, Uri Lerner, David M Broday, and Barak Fishbain. On the feasibility of measuring urban air pollution by wireless distributed sensor networks. *Science of The Total Environment*, 502:537–547, 2015.

[176] Arslan Munir, Prasanna Kansakar, and Samee U Khan. Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things. *IEEE Consumer Electronics Magazine*, 6(3):74–82, 2017.

[177] Asaf Nadler, Avi Aminov, and Asaf Shabtai. Detection of malicious and low throughput data exfiltration over the dns protocol. *Computers & Security*, 80:36–53, 2019.

[178] NDN-testbed. *Named Data Networking*, (accessed June, 2018).

[179] NetCraftsmen. USE DNS WHITELISTS TO STOP MALWARE IN ITS TRACKS. `https://netcraftsmen.com/use-dns-whitelists-to-stop-malware-in-its-tracks/`, 2022. [Online; accessed July-20-2022].

[180] PaloAlto Networks. Detecting DNS tunneling. `https://www.paloaltonetworks.com/resources/videos/lightboard-dns-security-service`, 2019. [Online; accessed October-01-2021].

[181] PaloAlto Networks. Stop attackers from using DNS against you. https://start.paloaltonetworks.com/protect-your-dns-traffic-against-threats.html, 2022. [Online; accessed June-01-2022].

[182] Joao C Neves and Hugo Proença. Dynamic camera scheduling for visual surveillance in crowded scenes using markov random fields. In *AVSS*, pages 1–6. IEEE, 2015.

[183] Nam T Nguyen, Svetha Venkatesh, Geoff West, and Hung H Bui. Multiple camera coordination in a surveillance system. *ACTA Automatica Sinica*, 29(3):408–422, 2003.

[184] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.

[185] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. Using of jaccard coefficient for keywords similarity. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2013.

[186] norton.com. What is a trojan downloader? https://us.norton.com/blog/malware/what-is-a-trojan-downloader, 2018.

[187] University of Southern California-Information Sciences Institute. Day in the life of the internet (ditl). https://www.impactcybertrust.org/dataset_view?idDataset=884, 2022.

[188] University of Southern California-Information Sciences Institute. Reverse dns. https://www.impactcybertrust.org/dataset_view?idDataset=702, 2022.

[189] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*.

[190] openaddresses.io. batch.openaddresses.io/data. {https://batch.openaddresses.io/data}, 2016. [Online; accessed December-1-2021].

[191] OpenAI. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt/, 2022. [Online; accessed Dec-12-2022].

[192] Carlos Ordonez. Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 12–19. ACM, 2003.

[193] PaloAlto. Strategically Aged Domain Detection: Capture APT Attacks With DNS Traffic Trends. https://unit42.paloaltonetworks.com/strategically-aged-domain-detection/, 2021. [Online; accessed December-15-2021].

[194] Ruchi Parikh and Kamalakar Karlapalem. Et: events from tweets. In *Proceedings of the 22nd international conference on world wide web*, pages 613–620. ACM, 2013.

[195] Unsang Park, Hyun-Cheol Choi, Anil K Jain, and Seong-Whan Lee. Face tracking and recognition at a distance: A coaxial and concentric ptz camera system. *IEEE transactions on information forensics and security*, 8(10):1665–1677, 2013.

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

[197] Vern Paxson, Mihai Christodorescu, Mobin Javed, Josyula Rao, Reiner Sailer, Douglas Lee Schales, Mark Stoecklin, Kurt Thomas, Wietse Venema, and Nicholas Weaver. Practical comprehensive bounds on surreptitious communication over {DNS}. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 17–32, 2013.

[198] Andy Penfold. How iot is reshaping the future of video surveillance. `{https://www.securityandsafetythings.com/insights/iot-reshaping-future-surveillance}`.

[199] Claudio Piciarelli, Lukas Esterle, Asif Khan, Bernhard Rinner, and Gian Luca Foresti. Dynamic reconfiguration in camera networks: A short survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(5):965–977, 2015.

[200] Raymond Pompon. Cybersecurity threats to the covid-19 vaccine. `https://www.f5.com/labs/articles/threat-intelligence/cybersecurity-threats-to-the-covid-19-vaccine`, 2021.

[201] Martin Porter. Porter Stemmer. `{https://tartarus.org/martin/PorterStemmer/}`, Jan, 2006. [Online; accessed Oct-11-2018].

[202] NEWYORK POST. *Harvey victims are using social media when 911 fails*, (accessed September, 2018).

[203] ONOS project. *ONOS SDN*, (accessed June, 2018).

[204] Faisal Z Qureshi and Demetri Terzopoulos. Planning ahead for ptz camera assignment and handoff. In *ICDSC*, pages 1–8. IEEE, 2009.

[205] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[206] Anand Rajaraman and Jeffrey David Ullman. *Data Mining*, page 1–17. Cambridge University Press, 2011.

[207] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[208] Philippe Remy. Name dataset. `https://github.com/philipperemy/name-dataset`, 2021.

[209] Sam Rhea. Announcing cloudflare's data loss prevention platform. https://blog.cloudflare.com/data-loss-prevention/, 2021.

[210] Paul Roberts. What's The Value of a Stolen Chest X-Ray? More Than You'd Think. {https://www.digitalguardian.com/blog/whats-value-stolen-chest-x-ray-more-youd-think}, 2021.

[211] Stefan Rudolph, Sarah Edenhofer, Sven Tomforde, and Jörg Hähner. Reinforcement learning for coverage optimization through ptz camera alignment in highly dynamic environments. In *Proceedings of the International Conference on Distributed Smart Cameras*, pages 1–6, 2014.

[212] Carlos Sampedro, Alejandro Rodriguez-Ramos, Hriday Bavle, Adrian Carrio, Paloma de la Puente, and Pascual Campoy. A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques. *Journal of Intelligent & Robotic Systems*, 95(2):601–627, 2019.

[213] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[214] Manos Schinas, Symeon Papadopoulos, Yiannis Kompatsiaris, and Pericles A Mitkas. Visual event summarization on social media using topic modelling and graph-based ranking algorithms. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 203–210. ACM, 2015.

[215] Stefan Schneider, Haydar Qarawlus, and Holger Karl. Distributed online service coordination using deep reinforcement learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 539–549. IEEE, 2021.

[216] securereading. How a hacker can infiltrate your network and what can be done about it? https://securereading.com/hacking-hacker-infiltrate-networks/, 2016.

[217] David A Shamma, Lyndon Kennedy, and Elizabeth F Churchill. Peaks and persistence: modeling the shape of microblog conversations. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 355–358. ACM, 2011.

[218] Huajie Shao, Dachun Sun, Shuochao Yao, Lu Su, Zhibo Wang, Dongxin Liu, Shengzhong Liu, Lance Kaplan, and Tarek Abdelzaher. Truth discovery with multimodal data in social sensing. *IEEE Transactions on Computers*, 2020.

[219] Navin K Sharma, David E Irwin, Prashant J Shenoy, and Michael Zink. Multisense: fine-grained multiplexing for steerable camera sensor networks. In *ACM conference on Multimedia systems*, 2011.

[220] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[221] M Hassan Shirali-Shahreza and Mohammad Shirali-Shahreza. Text steganography in chat. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–5. IEEE, 2007.

[222] Manmeet Singh, Maninder Singh, and Sanmeet Kaur. Ti-2016 dns dataset. https://dx.doi.org/10.21227/9ync-vv09, 2019.

[223] Nir Soft. Dns query sniffer. https://www.nirsoft.net/utils/dns_query_sniffer.html, 2013.

[224] GNU software. *wget*, (accessed June, 2018).

[225] Liming Song, Wenfu Wu, Junrong Guo, and Xiuhua Li. Survey on camera calibration technique. In *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 389–392. IEEE, 2013.

[226] Sony. Remotely controlled PTZ color video camera with IP streaming. {https://pro.sony/ue_US/products/ptz-network-cameras/srg-300se}.

[227] Sony. Visca command control. {https://aca.im/driver_docs/Sony/EVI-H100V-S-Tech-Manual.pdf}. [Online; accessed March-30-2020].

[228] Tim Soulo. 90.63% of Content Gets No Traffic From Google. And How to Be in the Other 9.37% [New Research for 2020]. https://ahrefs.com/blog/search-traffic-study/, 2022. [Online; accessed July-20-2022].

[229] Tim Stack. Internet of Things (IoT) Data Continues to Explode Exponentially. Who Is Using That Data and How? {https://blogs.cisco.com/datacenter/internet-of-things-iot-data-continues-to-explode-exponentially-who-is-using-that-d Feb 5, 2018. [Online; accessed Oct-11-2018].

[230] Wiktor Starzyk and Faisal Z Qureshi. Learning proactive control strategies for ptz cameras. In *2011 Fifth ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–6. IEEE, 2011.

[231] Jingwen Su and Hujun Yin. Efficient multi-objective gans for image restoration. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1855–1859. IEEE, 2021.

[232] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[233] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[234] Symantec. Sunburst: Supply Chain Attack Targets SolarWinds Users. https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/sunburst-supply-chain-attack-solarwinds, 2021. [Online; accessed June-15-2021].

[235] Georgia Tech. Gt malware passive dns data daily feed (07/01/2015 to 12/31/2017). https://www.impactcybertrust.org/dataset_view?idDataset=520, 2015.

[236] Ars Technica. *Ars Technica*, (accessed August, 2018).

[237] Telegraph. Moscow says Twitter ready to store data of users on Russian servers despite concerns over surveillance . {https://www.telegraph.co.uk/news/2017/11/08/moscow-says-twitter-ready-store-data-users-russian-servers-despite/}, Nov, 2017. [Online; accessed Oct-11-2018].

[238] the Broadcast Bridge. The rise of the ptz camera. {https://www.thebroadcastbridge.com/content/entry/12065/the-rise-of-the-ptz-camera}, Dec, 2019. [Online; accessed March-30-2020].

[239] Iris Tien, Aibek Musaev, David Benas, Ameya Ghadi, Seymour Goodman, and Calton Pu. Detection of damage and failure events of critical public infrastructure using social sensor big data. In *IoTBD*, 2016.

[240] Rohit Kumar Tiwari and Gyanendra K Verma. A computer vision based framework for visual gun detection using harris interest point detector. *Procedia Computer Science*, 54:703–712, 2015.

[241] NLTK Tokenizer. NLTK 3.3 documentation. {https://www.nltk.org/api/nltk.tokenize.html}. [Online; accessed Oct-11-2018].

[242] Tornado. *Tornado web server*, (accessed June, 2018).

[243] F. Toutounchi and E. Izquierdo. Enhancing digital zoom in mobile phone cameras by low complexity super-resolution. In *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 01–06, 2018.

[244] Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. *Advances in Neural Information Processing Systems*, 32, 2019.

[245] Zongjie Tu and Prabir Bhattacharya. Game-theoretic surveillance over arbitrary floor plan using a video camera network. *Signal, Image and Video Processing*, 7(4):705–721, 2013.

[246] Burak Uzkent and Stefano Ermon. Learning when and where to zoom with deep reinforcement learning. In *CVPR*, pages 12345–12354, 2020.

[247] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[248] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[249] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. Gated siamese convolutional neural network architecture for human re-identification. In *ECCV*, pages 791–808. Springer, 2016.

[250] Varonis. What is DNS tunneling? A detection guide. `https://www.varonis.com/blog/dns-tunneling/`, 2020. [Online; accessed Mar-01-2022].

[251] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[252] The Verge. The us, like china, has about one surveillance camera for every four people, says report. `{https://www.theverge.com/2019/12/9/21002515/surveillance-cameras-globally-us-china-amount-citizens}`, Dec, 2019. [Online; accessed March-30-2020].

[253] Maximilian Walther and Michael Kaisser. Geo-spatial event detection in the twitter stream. In *European conference on information retrieval*, pages 356–367. Springer, 2013.

[254] Daimeng Wang, Ajaya Neupane, Zhiyun Qian, Nael B Abu-Ghazaleh, Srikanth V Krishnamurthy, Edward JM Colbert, and Paul Yu. Unveiling your keystrokes: A cache-based side-channel attack on graphics libraries. In *NDSS*, 2019.

[255] Dong Wang, Tarek Abdelzaher, and Lance Kaplan. *Social sensing: building reliable systems on unreliable data*. Morgan Kaufmann, 2015.

[256] Dong Wang, Md Tanvir Amin, Shen Li, Tarek Abdelzaher, Lance Kaplan, Siyu Gu, Chenji Pan, Hengchang Liu, Charu C Aggarwal, Raghu Ganti, et al. Using humans as sensors: an estimation-theoretic perspective. In *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*, pages 35–46. IEEE, 2014.

[257] Dong Wang, Lance Kaplan, Hieu Le, and Tarek Abdelzaher. On truth discovery in social sensing: A maximum likelihood estimation approach. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 233–244. ACM, 2012.

[258] Qi Wang, Jianmin Liu, Katia Jaffrès-Runser, Yongqing Wang, Chentao He, Cunzhuang Liu, and Yongjun Xu. Incdeep: intelligent network coding with deep reinforcement learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[259] Shiguang Wang, Prasanna Giridhar, Hongwei Wang, Lance Kaplan, Tien Pham, Aylin Yener, and Tarek Abdelzaher. Storyline: Unsupervised geo-event demultiplexing in social spaces without location information. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 83–93. ACM, 2017.

[260] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[261] Kazufumi Watanabe, Masanao Ochi, Makoto Okabe, and Rikio Onai. Jasmine: a real-time local-event detection system based on geolocation information propagated to microblogs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2541–2544. ACM, 2011.

[262] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[263] Tianshu Wei, Yanzhi Wang, and Qi Zhu. Deep reinforcement learning for building hvac control. In *DAC*, 2017.

[264] Ziyan Wu and Richard J Radke. Keeping a pan-tilt-zoom camera calibrated. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1994–2007, 2012.

[265] www.statista.com. Number of target stores in the united states from financial year 2006 to 2022. `https://www.statista.com/statistics/255965/total-number-of-target-stores-in-north-america/`, 2013.

[266] www.trendmicro.com. Targeted Attacks. `{https://www.trendmicro.com/vinfo/us/security/definition/targeted-attacks}`, 2020. [Online; accessed June-1-2022].

[267] xairy. Linux kernel exploitation. `https://github.com/xairy/linux-kernel-exploitation`.

[268] Chaocan Xiang, Panlong Yang, Chang Tian, Haibin Cai, and Yunhao Liu. Calibrate without calibrating: An iterative approach in participatory sensing network. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):351–361, 2014.

[269] Yuanlu Xu, Liang Lin, Wei-Shi Zheng, and Xiaobai Liu. Human re-identification by matching compositional template with cluster sampling. In *proceedings of the IEEE International Conference on Computer Vision*, pages 3152–3159, 2013.

[270] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.

[271] Yi Yao, Chung-Hao Chen, Andreas Koschan, and Mongi Abidi. Adaptive online camera coordination for multi-camera multi-target surveillance. *Computer Vision and Image Understanding*, 114(4):463–474, 2010.

[272] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM Sigmod record*, 31(3):9–18, 2002.

[273] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE international conference on computer vision*, pages 2849–2857, 2017.

[274] Jinsung Yoon, William R Zame, and Mihaela Van Der Schaar. Deep sensing: Active sensing using multi-directional recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[275] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 340–349, 2018.

[276] Jiacheng Zhang, Li Yang, Shui Yu, and Jianfeng Ma. A dns tunneling detection method based on deep learning models to prevent data exfiltration. In *Network and System Security: 13th International Conference, NSS 2019, Sapporo, Japan, December 15–18, 2019, Proceedings 13*, pages 520–535. Springer, 2019.

[277] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[278] Guoqiang Zhong, Wei Gao, Yongbin Liu, Youzhao Yang, Da-Han Wang, and Kaizhu Huang. Generative adversarial networks with decoder–encoder output noises. *Neural Networks*, 127:19–28, 2020.

[279] Shi Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5-6):790–798, 2005.

[280] Kaiyang Zhou and Tao Xiang. Torchreid: A library for deep learning person reidentification in pytorch. *arXiv preprint arXiv:1910.10093*, 2019.

[281] Xiangmin Zhou and Lei Chen. Event detection over twitter social media streams. *The VLDB Journal—The International Journal on Very Large Data Bases*, 23(3):381–400, 2014.

[282] Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *International Workshop on Internet and Network Economics*, pages 566–576. Springer, 2008.

[283] Bolor-Erdene Zolbayar, Ryan Sheatsley, Patrick McDaniel, Michael J Weisman, Sencun Zhu, Shitong Zhu, and Srikanth Krishnamurthy. Generating practical adversarial network traffic flows using nidsgan. *arXiv preprint arXiv:2203.06694*, 2022.