

**From the holdings of the Department of Special Collections , Princeton University Library,
Princeton, NJ**

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyright law. We do not charge any permission or usage fees for the publication of images of material in our collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff. For more information regarding copyright, credit, and citation please review the information on the Copyright, Credit and Citation Guidelines page of our website.

<https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines>. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. <https://library.princeton.edu/special-collections/ask-us>

Master's Theses, 2022-2026

Computer Science, 2023

2024

SketchProbe: Discovering Vulnerabilities in Sketch-based Applications with Reinforcement Learning

Luo, Wei

Princeton, NJ : Princeton University

<http://arks.princeton.edu/ark:/88435/dsp013j3335631>

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. <https://www.copyright.gov/title17/title17.pdf>

SKETCHPROBE: DISCOVERING
VULNERABILITIES IN SKETCH-BASED
APPLICATIONS WITH REINFORCEMENT
LEARNING

WEI LUO

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF MASTER OF SCIENCE IN ENGINEERING

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE
ADVISER: PROFESSOR MARIA APOSTOLAKI

MAY 2024

© Copyright by Wei Luo, 2024.

All rights reserved.

Abstract

Sketches are approximate data structures critical to network traffic monitoring. They are specifically designed for environments constrained by memory, enabling the maintenance of accurate statistics within a compact space. This efficiency comes at the cost of estimation error that varies based on the workload encountered by the sketch. Traditionally, sketch-based applications are optimized for average traffic scenarios, and this assumption of standard patterns of network traffic introduces vulnerabilities, as these applications may not be prepared for atypical or adversarial traffic patterns. Manual testing for these vulnerabilities is time-intensive and often impractical due to the vast range of possible traffic scenarios. In this paper, we introduce **SketchProbe**, a novel system leveraging Reinforcement Learning (RL) to identify adversarial traffic patterns in sketch-based applications. Unlike manual identification, **SketchProbe** automates the discovery process, ensuring a thorough and efficient exploration of potential failure modes. Our evaluation indicates that **SketchProbe** effectively identifies critical scenarios that significantly impact the performance of sketch-based applications, which is valuable for improving the robustness of such applications.

Acknowledgements

To Professor Maria Apostolaki, thank you for being such a fantastic advisor. Thank you for always being helpful and responsive. I'm so glad I've had the opportunity to work with you this year and I'm incredibly grateful for all you've taught me.

To Mary Hogan, thank you for working with me on this project all the way from start to finish. I learned so much knowledge from you that I was previously unaware of. Thank you for always being patient in answering my questions and giving valuable advice regarding the project.

To Professor Benjamin Eysenbach, thank you for being my thesis reader, always appearing enthusiastic at classes and my presentations. Thank you for providing timely feedback on my thesis. I have gained a lot from it.

To my parents, thank you for your unconditional love and support throughout my time here at Princeton and throughout my life.

To my parents.

Contents

Abstract	3
Acknowledgements	4
1 Introduction	8
2 Background	12
2.1 Programmable Switches	12
2.2 Approximate Data Structures	13
2.3 Reinforcement Learning	17
2.4 Motivation	20
3 SketchProbe Design	23
3.1 Threat Model	23
3.2 RL Model	26
3.3 System Optimization	30
4 Implementation	32
4.1 Network Simulator	32
4.2 RL Agent	33
5 Evaluation	36
5.1 Methodology	36
5.2 Results	39

6	Related Work	44
6.1	Sketching Data Structures	44
6.2	RL in Networking	45
7	Concluding Remarks	47
7.1	Conclusion	47
7.2	Future Work	48
	Bibliography	50

Chapter 1

Introduction

In the realm of network management and security, efficiently and accurately monitoring traffic at high speeds is essential. Programmable switches have revolutionized how networks handle traffic, introducing new levels of programmability while maintaining line rate processing speeds.

Despite these advancements, the high operating speeds of these devices impose significant processing and resource constraints. The swift pace at which data must be processed often limits the amount of memory and computation available for each packet, presenting a formidable challenge in sustaining thorough and precise network monitoring. To navigate these limitations, approximate data structures have been employed, striking a balance between resource efficiency and accuracy. These data structures are critical for maintaining essential network statistics, which provide insights into traffic patterns, peak usage times, and potential security threats. By producing estimates with acceptable error margins and utilizing minimal memory, these structures ensure effective operation within the constrained environment of programmable switches, which is crucial in high-speed network settings.

Approximate data structures, such as sketches and hash tables, are optimized to handle typical or expected traffic patterns efficiently. Normal traffic patterns generally

display a predictable and relatively uniform distribution of data, allowing these structures to estimate network statistics with minimal error. However, their performance can significantly degrade under unusual or adversarial traffic patterns. Unusual or abnormal traffic patterns can include sudden spikes in activity, highly skewed distributions of requests (where a few sources generate the majority of traffic), or deliberately crafted sequences of data designed to disrupt normal operations. These conditions can lead to hash collisions—a scenario where different data inputs are mapped to the same index in the structure—resulting in inaccuracies and compromised data integrity.

Adversaries might exploit these vulnerabilities by crafting traffic specifically to induce hash collisions. For instance, by sending a flood of packets with spoofed IP addresses that hash to the same index, attackers can significantly amplify estimation errors. This manipulation can distort the perceived state of the network, making it difficult for network managers to detect or mitigate issues effectively, leading to potential security breaches or service disruptions. Such tactics underscore the importance of designing approximate data structures that are robust not only under standard conditions but also in the face of these adversarial scenarios.

In the work, we present **SketchProbe**, a system that aims to enhance the resilience and security of network monitoring systems that utilize such approximate data structures, where the system can maintain accurate functionality and performance despite unexpected or malicious traffic patterns and can protect against intentional manipulations and attacks that aim to compromise data integrity or disrupt network operations. Given the complex variety of potential traffic patterns, manually identifying and testing adversarial conditions is both impractical and inefficient. To overcome this challenge, **SketchProbe** leverages Reinforcement Learning (RL) to automate this process, significantly enhancing the security and performance of network monitoring systems.

SketchProbe was designed to tackle the challenge of identifying adversarial traffic patterns by utilizing the following strategies:

- (1) Automated Exploration: **SketchProbe** automates the search process, efficiently exploring a wide range of traffic scenarios beyond the capabilities of manual methods.
- (2) Time and Resource Efficiency: By using RL, **SketchProbe** minimizes the need for extensive computational resources and the extensive trial and error associated with manual testing, streamlining the discovery process.
- (3) Enhanced Robustness: By pinpointing critical scenarios that could impact sketch-based applications, **SketchProbe** aids in developing more resilient network traffic monitoring solutions capable of withstanding adversarial conditions.

To validate that RL is a viable strategy for this network challenge, we conducted a simplified test in an isolated network environment free from background traffic. This setting allowed us to focus on the interactions between the RL agent and the sketch-based applications, providing a proof of concept for **SketchProbe**'s capabilities.

Our evaluation in scenarios with realistic background traffic further demonstrates the efficacy of **SketchProbe**. The findings indicate that **SketchProbe**'s attack methodology induces a higher error rate compared to scenarios with no attack or random attack, highlighting its effectiveness. The flexibility of **SketchProbe** is shown through its successful implementation using various RL techniques, each exhibiting distinct performance levels. Additionally, the robustness of the attacks is affirmed across different traffic distributions, confirming **SketchProbe**'s potential to disrupt network traffic monitoring in diverse network conditions.

In summary, the contributions of **SketchProbe** are significant:

- (1) We designed an RL-based attack generation framework specifically for sketch-based applications.

- (2) We implemented **SketchProbe**, providing a practical tool for identifying critical scenarios and generating adversarial traffic patterns.
- (3) We evaluated **SketchProbe** under various conditions, demonstrating its effectiveness and value in enhancing the robustness of approximate data structures for network programmers.

Chapter 2

Background

2.1 Programmable Switches

Programmable switches [4, 20] empower network engineers to customize traffic handling and optimization, enhancing network flexibility, performance, and security. For instance, the Intel Tofino showcases modern programmable switches' capabilities by offering a high-performance, protocol-independent architecture. Figure 2.1 is an example of a Protocol Independent Switch Architecture (PISA) [14]. This allows users to define and implement complex network functions directly in hardware, from basic routing to advanced traffic management and security monitoring.

However, despite their advanced features, programmable switches face stringent processing and resource limitations due to their pipeline architecture:

- (1) They consist of multiple stages, each performing specific tasks like header processing and forwarding decision-making.
- (2) Limited onboard memory and registers are available for storing persistent state information, such as flow table entries and statistics counters.
- (3) Each stage has a finite number of register accesses, constraining the complexity of packet processing tasks.

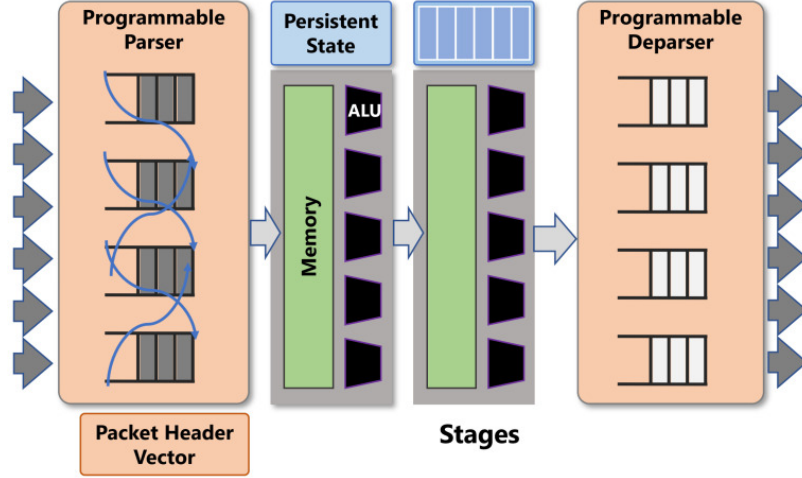


Figure 2.1: An example of a PISA. Adapted from [14].

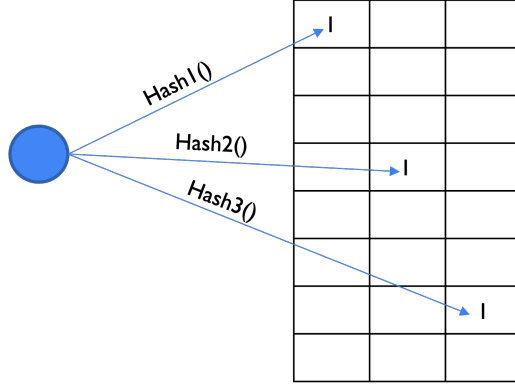
As network environments evolve, understanding and leveraging programmable switches’ capabilities become increasingly critical for enhancing network operations’ efficacy and security.

2.2 Approximate Data Structures

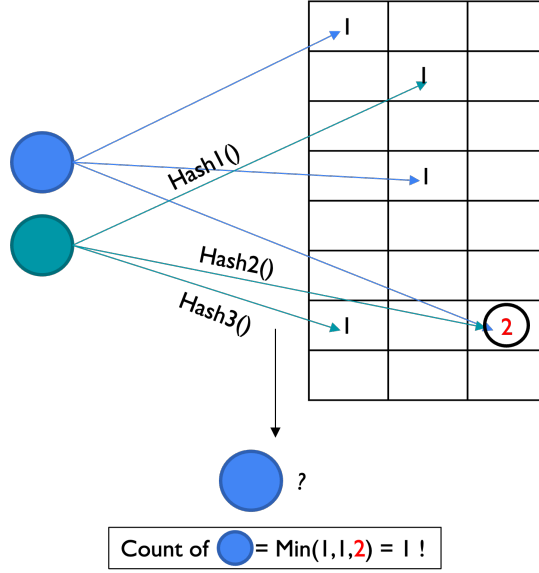
Definition. An approximate data structure is a type of algorithmic strategy designed to provide quick and space-efficient ways to handle large datasets where exact solutions are either impossible or impractical due to resource constraints. They are particularly useful for big data applications and are engineered to deliver high-performance computations while balancing the trade-off between accuracy, speed, and memory usage. Approximate data structures trade error of approximation for faster operation, leading to theoretical and practical speedups for a wide variety of algorithms [22]. Approximate data structures are commonly used for tasks in networking environments, such as frequency estimation, heavy hitter identification, cardinality estimation, etc. Using approximate data structures in networking environments has benefits of scalability, real-time processing, and resource efficiency. A sketch is an example of an approximate data structure widely used in computer networks that

summarizes network statistics by hashing the packet into indexes, and updating a small array of counters based on the indexes. They can be implemented in the data planes of switches and can operate at line-rate. They can be used to provide data such as packet frequency, and are often used in anomaly or adversary detection [26] and heavy hitter detection. Outside of the network world, they are also used in databases and password security [9]. Other examples of such data structures include Bloom Filters [3], Count Sketches [5], and HyperLogLog [11].

Security Implications. Approximate data structures utilize hashing algorithms to efficiently condense a large set of inputs into a more compact representation, balancing memory and computational resource constraints [22]. Despite their utility, these structures are not perfect by design; they are vulnerable when there is adversarial traffic since approximate data structures are optimized for average traffic. The accuracy of measurements of the approximate data structures fluctuates based on the traffic [7]. The Count-Min Sketch (CMS) [8] is a data structure designed for approximate frequency counting in streams of data. It consists of a two-dimensional array of counters, with each row corresponding to a hash function and each column representing a counter. When an element is hashed using multiple hash functions, its count is incremented in corresponding counters across all rows. This process ensures that each element is mapped to multiple counters, allowing for more accurate frequency estimation. By querying the minimum count across all corresponding counters for a given element, the CMS provides an approximate frequency count with controlled error guarantees. Overall, the CMS operates by incrementing counts in multiple counters based on hash functions and then querying the minimum count to estimate the frequency of elements in the stream. Figure 2.2 shows a simple case for querying packet count from a CMS. We propose to investigate and develop automated attacks on the CMS, specifically focusing on generating traffic patterns or workloads that maximize estimation errors by inducing collisions. These insights will assist network



(a) Blue Packet with Hash Functions.



(b) Teal Packet with Hash Functions, and Query the Count of the Blue Packet.

Figure 2.2: An example of a CMS with 3 columns and 8 rows.

researchers in creating more robust sketches that are less sensitive to such adversarial manipulations and better suited to handle worst-case scenarios [18].

Rotating Count-Min Sketch. The Rotating Count-Min Sketch (RCMS) introduces a dynamic version of the standard CMS, incorporating multiple sketches whose roles are systematically rotated at regular intervals. As depicted in Figure 2.3, the rotation process involves designating one sketch for writing incoming data while another is simultaneously tasked with cleaning. Cleaning involves removing outdated data to make room for new measurements. Every time a new packet arrives, the RCMS

learns the most recent traffic patterns by updating counters in the sketch designated for writing, while simultaneously cleaning outdated counters in the other sketch. This cleaning process is crucial due to the limitation on memory accesses in the switch. Programmable switches can only access one register or memory address per row in the array of counters. Thus, RCMS can only clean one slot per row at a time. Moreover, cleaning can only occur when a packet is received because switches cannot trigger events without packet transmissions. Therefore, RCMS adapts to traffic flow, ensuring that the data structure is modified only when necessary. By transitioning between sketches based on traffic flow instead of relying on automatic time-triggered cleaning processes, RCMS facilitates continuous and uninterrupted monitoring.

Rotating Count-min Sketch (RCMS)

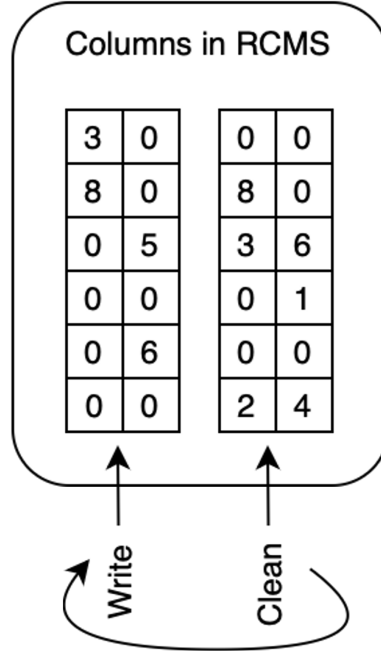


Figure 2.3: An example of a RCMS with 2 rotating sketches.

This method ensures that the data structure remains responsive to network traffic dynamics while evenly distributing the load across multiple sketches. In our paper, we focus on attacking the RCMS structure due to its method of periodically expunging

old data. This introduces a continually evolving challenge in maintaining the integrity of the captured data. By manipulating the traffic interacting with this system, we aim to explore and expose potential vulnerabilities in how RCMS manages and transitions its data.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning approach where an agent learns to make decisions by trying different actions and observing the outcomes [19]. In this approach, the agent interacts with an environment by choosing actions from a set of possible options, and it receives feedback as rewards or penalties based on the system's state. The goal of the agent is to maximize the total reward over time by learning to choose actions that yield higher rewards.

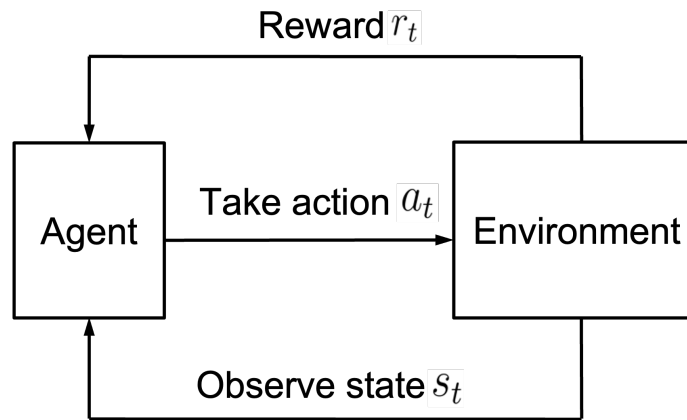


Figure 2.4: Reinforcement learning workflow.

In essence, RL considers a general setting in which an agent interacts with an environment. At each time step t , the agent observes some state s_t , and chooses an action a_t . After applying the action, the state of the environment transitions to s_{t+1} and the agent receives a reward r_t . The goal of learning is to maximize the expected

cumulative discounted reward:

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$$

is a factor discounting future rewards. This dynamic is depicted in Figure 2.4.

Reinforcement Learning Algorithm. Reinforcement learning algorithms can employ either a value-based approach or a policy-based approach. There are various algorithmic implementations for a reinforcement learning agent, which is referred to as the RL Agent. Value-based RL algorithms estimate the value of state-action pairs, representing the expected cumulative reward from a given state onwards. Methods like Q-learning and DQN learn Q-values to guide action selection by maximizing expected returns [17]. Policy-based RL algorithms directly learn the agent’s strategy for action selection given states, optimizing the policy to maximize cumulative rewards. Examples include REINFORCE and PPO, which parameterize and optimize the policy directly [10]. The RL Agent makes decisions within the environment. Here, we provide a brief overview of four reinforcement learning algorithms.

- (1) **Tabular Reinforcement Learning (Tabular RL).** Tabular RL is a value-based method where the agent’s knowledge about the environment is stored in a table [27]. Each entry in this table represents a state-action pair, with the values indicating the estimated reward for taking a specific action in a given state. This approach is straightforward and often used in environments with a discrete and small number of states and actions, allowing the agent to learn optimal decision-making by updating the table entries based on experiences (rewards and transitions) gathered through interaction with the environment. As the agent explores the environment, it refines these values, aiming to maximize cumulative rewards over time.
- (2) **Basic Deep Q-Network (DQN).** DQN is a value-based method, where the network is trained using a reinforcement learning algorithm that iteratively up-

dates the weights of the network to maximize the expected cumulative future rewards. This is achieved by minimizing the difference between the predicted Q-values, representing the expected cumulative reward that an agent can obtain by taking a particular action in a given state and then following a specific policy thereafter, and the target Q-values, which are calculated using the Bellman equation expressed mathematically as $Q(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a') | s, a]$, where $Q(s, a)$ is the value of taking action a in state s , r is the immediate reward, γ is the discount factor, s' is the next state, and a' is the next action. The learning process involves collecting experiences (state, action, reward, next state), storing them in a replay buffer, and using batches of these experiences to perform updates, thereby improving the policy iteratively. This architecture, although simple, can effectively learn policies in environments with relatively low complexity or when only a basic approximation of the Q-function is sufficient [23].

- (3) **Actor Critic (A2C).** Actor-critic algorithms combine both value-based and policy-based methods, by employing an actor which learns a policy, and a critic which evaluates the outcomes of the actor's actions. The actor is responsible for selecting actions based on the current policy. It maps states to actions, trying to find the policy that maximizes the cumulative reward. The policy is often represented as a probability distribution over actions that the actor outputs, allowing the model to explore the action space effectively. The critic assesses how good the action taken by the actor was and predicts future rewards. The critic's job is to evaluate the current policy by computing the value function, which estimates how good it is to be in a given state or how good an action taken in a given state is expected to be. This value function helps guide the actor toward more rewarding actions [13]. In the Actor-Critic method, both the actor and the critic are updated simultaneously during training. The critic

updates its value function based on the difference between the predicted rewards and the actual rewards received (the temporal difference error), which helps it to better evaluate the actions chosen by the actor. The actor updates its policy based on the feedback from the critic, adjusting towards actions that lead to higher value states as indicated by the critic. This dual approach allows the Actor-Critic method to be more stable than policy-based methods alone, as the critic’s evaluation helps to reduce the variance in updates. At the same time, it can learn more efficiently than value-based methods by using the gradient of the policy for updates, facilitating exploration of the action space.

- (4) **Proximal Policy Optimization (PPO).** PPO is a policy gradient reinforcement learning method, and is specifically designed to optimize policies in continuous control and high-dimensional action spaces. A particular advantage of PPO is that it uses a clip function to ensure that updates to the policy are within a specified, limited range, preventing too-large policy updates that can destabilize the learning process and lead to poor performance [25]. The PPO algorithm collects trajectories, computes advantages for each state-action pair, and updates the policy using a clipped surrogate objective. It improves sample efficiency by estimating state values without explicit exploration and is known for its computational efficiency compared to other policy gradient methods.

2.4 Motivation

The primary motivation behind the development of **SketchProbe** arises from the critical need to enhance the resilience and security of network monitoring systems, particularly those that utilize approximate data structures like sketches. Given the vast and complex landscape of potential traffic patterns, manual identification and testing of the adversarial conditions are not only impractical but also inefficient. This challenge

underscores the necessity for an automated, intelligent system capable of understanding and adapting to the dynamics of network traffic with minimal human intervention. **SketchProbe** leverages the power of RL to automate the exploration and identification of these critical scenarios, thus providing a robust solution that enhances the performance and security of network monitoring systems. By focusing on this aspect, **SketchProbe** aims to equip network programmers with a more resilient toolset, better prepared to handle the dynamic and unpredictable nature of network environments, thereby safeguarding the infrastructure against potential vulnerabilities.

To demonstrate that RL is a viable strategy for addressing our network challenge, we explore a simplified scenario. In this scenario, the RL agent’s action space is binary: it can either send a burst of 500 packets or send none (0 packets). This experiment is conducted in an isolated network environment free from background traffic, which eliminates external disturbances and allows for a focused analysis of the interactions between the RL agent and the RCMS. The primary goal of the RL agent in this context is to strategically generate and transmit packets to maximize hash collisions within the RCMS. The agent is programmed to adopt a specific “send (1) - not send (0) - send (1) - ...” alternating pattern, particularly effective due to the RCMS’s cleaning mechanism, which tends to amplify the resultant error rate when faced with this input pattern. This is because when no packets are sent, the RCMS cannot perform its cleaning operation, and the sketch counters will not reset, we’ll observe higher error rates due to the accumulation of stale information. The reward function employed quantifies the impact of the agent’s actions by calculating the maximum error observed in the RCMS over several rounds, effectively measuring the degradation in the sketch’s accuracy. The outcomes of this experiment reveal that the RL agent consistently converges to the optimal “send (1) - not send (0) - send (1) - ...” pattern, as illustrated in Figure 2.5. This behavior demonstrates the agent’s capability to adapt and strategically manipulate the RCMS to induce the highest

possible error rate, surpassing other potential patterns, such as “send (1) - send (1) - not send (0) - ...”, “send (1) - send (1) - send (1) - ...”, etc.. This minimal case scenario acts as a proof of concept, underscoring the potential of RL to detect and exploit operational vulnerabilities within sketch-based network monitoring systems.

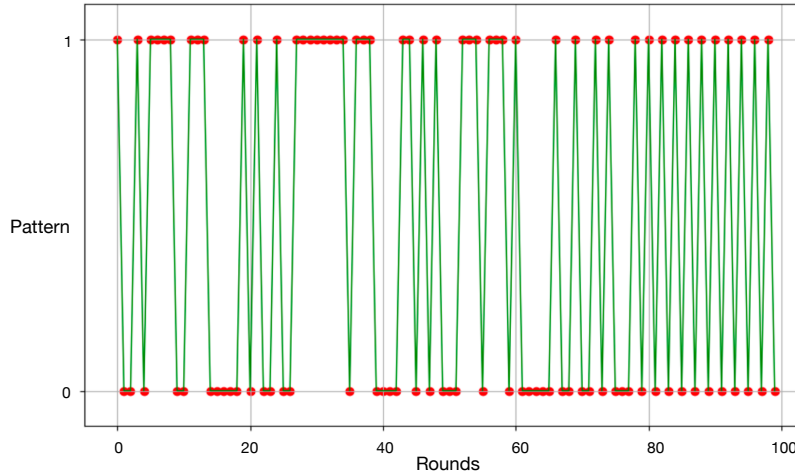


Figure 2.5: The graph illustrates the convergence to the “send (1) - not send (0) - send (1) - ...” pattern for the minimal case scenario. Each point on the graph represents the observed action in the RCMS over 100 rounds of training. Convergence is identified by the consistent adoption of the desired pattern by the RL agent, particularly notable when it starts consistently choosing the 1-0-1 pattern after around 80 rounds.

Chapter 3

SketchProbe Design

In this section, we outline the design of the **SketchProbe** system, focusing on the underlying threat model, the architecture of the RL-based attacker agent, and specific parameters and settings that influence the attacker’s effectiveness and explore how the system can be adapted to handle various attack scenarios effectively.

3.1 Threat Model

We focus on exploiting vulnerabilities in sketch-based applications. The RL agent’s primary objective is to compromise the accuracy of the CMS by generating traffic patterns that maximize hash collisions and degrade the effectiveness of this security mechanism. The threat model [28] assumes that the adversarial agent has a general understanding of the CMS structure, but lacks detailed knowledge of its internal configurations, such as specific flow details or hash functions. This limitation necessitates the development of general strategies that can systematically weaken the CMS’s defenses.

To rigorously test and validate our approach, we define specific experimental conditions and assumptions:

- (1) **Maximum Packet Limit:** The attacker can choose to send up to 500 packets each round, with the distribution being either skewed based on Zipfian distribution [24] or random.
- (2) **Consistent Network Conditions:** Both training and testing of the RL agent occur under similar conditions, with a constant sketch size and randomly generated background traffic to simulate a stable network environment.
- (3) **Scenarios for Evaluation:** We analyze two distinct cases:
 - (a) **Basic scenario with no background traffic:** Here, the RL agent operates in an isolated environment, allowing us to assess its effectiveness in directly attacking the CMS without external traffic interference, as shown in Figure 3.1.
 - (b) **Realistic scenario with background traffic:** This scenario introduces background traffic, providing a more complex and realistic setting that tests the agent’s ability to craft subtle and hard-to-detect attacks, as shown in Figure 3.2.

We fix the sketch size throughout our experiments since changing the sketch size would necessitate altering the action space accordingly. The primary indicators of success in our study are the consistent and significant disruption of the CMS functionality and the RL agent’s ability to adapt to various network conditions while sustaining effective attack strategies. This approach not only highlights the potential risks posed by intelligent adversaries but also underscores the need for robust defensive mechanisms in network security frameworks.

Attack Assumptions. Instead of blindly attacking the CMS, we consider a scenario where the attacker has access to the CMS’s count each time they manipulate network traffic. This access grants the attacker information about the counts of every distinct

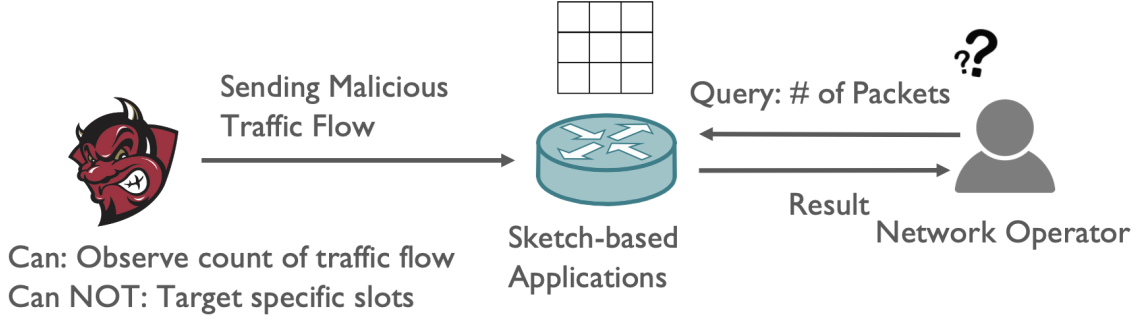


Figure 3.1: No background traffic threat model.

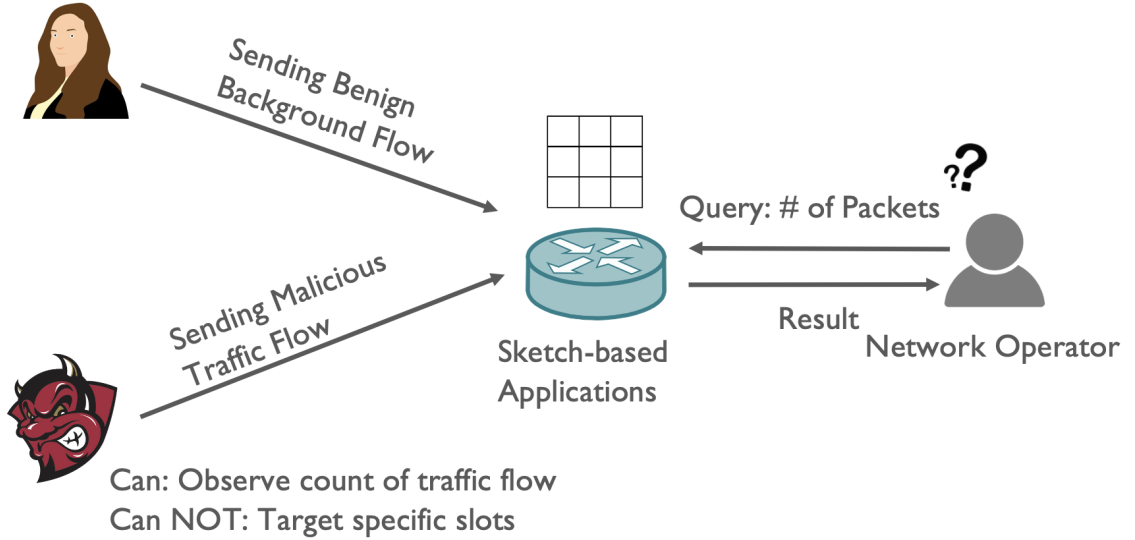


Figure 3.2: Realistic background traffic threat model.

value recorded by the CMS. This assumption is justified by the need to provide feedback to the agent in our experimental setup, ensuring that the attacker can make informed decisions based on the current state of the CMS.

Furthermore, the attacker’s capability to send traffic is limited to a predetermined number of packets. This constraint is designed to limit the action space of the RL

agent to achieve a more efficient attack. Beyond the specified insights into the CMS contents, the attacker lacks detailed information regarding other crucial aspects of the network’s defense system. Specifically, the attacker does not know the size of the sketch, the patterns of background traffic. This lack of comprehensive knowledge challenges the attacker to devise strategies that are both effective and stealthy under these constrained conditions.

3.2 RL Model

Our RL framework, depicted in the system overview diagram Figure 3.3, consists of an environment where the agent interacts. This environment provides a state from which the agent can perform actions based on a policy and receives feedback through a reward function. In the diagram, B_t represents the remaining packet budget after each action, A_t signifies the action taken by the agent at each time step t , and E_t denotes the error rate observed by the agent at time step t .

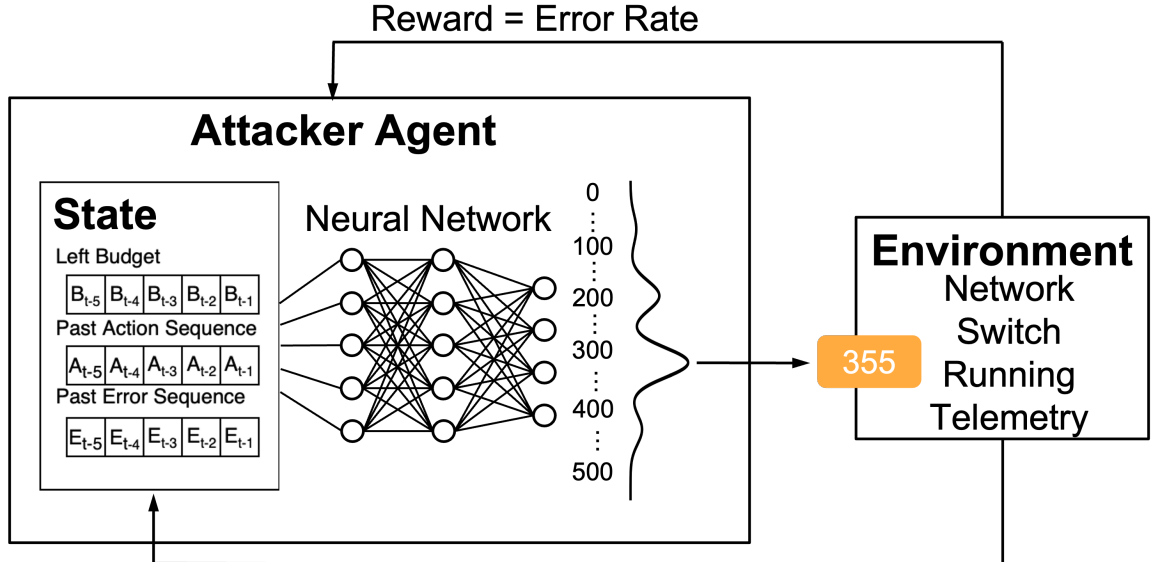


Figure 3.3: Reinforcement learning system overview.

Environment. The RL environment emulates a network switch running a telemetry application using rotating sketches, and it manages packet transmissions. The simulation of this environment involves actions such as sending packets, which can potentially alter network behavior by setting bits in a measurement application and thereby impacting the measurement of packet transmissions. This simulation effectively manages these dynamics by adjusting the internal state in response to the actions taken and updating operational parameters like budget, ensuring a realistic emulation of network operations.

State Representation. In the SketchProbe architecture, the state representation of the RL agent is designed to adapt according to the presence or absence of background traffic, providing a nuanced approach to attack strategy optimization.

- **State Representation Without Background Traffic:** When there is no background traffic, the state of the RL agent is represented by a 10-dimensional vector. This vector is equally divided between the past five action sequences and the past five corresponding error sequences, giving the agent historical context to inform future decisions. Each state vector element captures crucial aspects of the agent’s recent interactions with the network:

$$S = (A_1, A_2, \dots, A_5, E_1, E_2, \dots, E_5) \quad (3.1)$$

where A_i denotes the number of packets the agent decided to send in each of the previous five windows, and E_i corresponds to the error rates associated with these actions. This state structure allows the RL agent to track and analyze the immediate impact of its actions on network performance, facilitating a strategic response to evolving network conditions.

- **State Representation With Background Traffic:** In scenarios with background traffic, the state expands to a 15-dimensional vector to accommodate

additional complexities introduced by external traffic flows. This expanded state includes the same ten dimensions as the no-background scenario, with an additional five dimensions representing the remaining packet budget. The packet budget is defined as the number of packets that can still be dispatched within a total limit of 500 packets, enabling the agent to consider its capacity to influence the network further within the current operational constraints:

$$S = (B_1, B_2, \dots, B_5, A_1, A_2, \dots, A_5, E_1, E_2, \dots, E_5) \quad (3.2)$$

where B_i tracks the remaining packet budget after each action, reflecting the dynamic allocation of resources in response to both the agent’s activities and the background traffic conditions. This extended state space not only enhances the agent’s ability to make informed decisions based on a comprehensive view of the network environment but also supports the development of more sophisticated and realistic attack strategies that leverage fluctuations in network traffic.

These tailored state representations ensure that the RL agent can effectively adapt its strategies to different network conditions, optimizing its performance whether operating in isolation or in a complex network environment. Specifically, when there is background traffic, adding the budget sequence allows the agent to leverage the workload of current background traffic, because the agent will likely send the max each time without the budget as the action of sending the maximum number of packets results in the highest error. This also implies a more realistic attack since it will not always be incentive to send the max possible number of packets.

Action. The action space for the RL agent in **SketchProbe** is designed as a discrete set, comprising 501 distinct actions. These actions range from sending 0 to 500 packets per decision point. This detailed granularity enables the agent to make precise

decisions on the volume of traffic to inject at any given moment, but the discretion of the action space also make this decision fast enough to facilitate a finely tuned approach. By offering a wide range of packet-sending options, the agent can strategically balance the intensity of its actions—choosing not to send packets, sending a minimal number to test system responses, or sending large bursts to stress the network monitors.

Reward. The reward function quantifies the effectiveness of the RL agent’s actions in disrupting the sketch’s ability to accurately monitor traffic flows. It is mathematically expressed as:

$$R = \frac{1}{N} \sum_{i=1}^N \frac{|c_i - \hat{c}_i|}{c_i} \quad (3.3)$$

where $i = 1, \dots, N$ represents the indexes of the flows within the background traffic, c_i denotes the true count of packets for each flow, and $\hat{c}_i$ is the estimated count as determined by the network’s monitoring system. This formula calculates the average relative error across all monitored flows. The reward structure encourages strategies that are effective across the entirety of the network traffic. This approach ensures that the RL agent’s strategies are robust.

Policy. For our policy implementation, we primarily utilize Deep Q-Learning (DQL) to accommodate the complex and dynamic nature of network environments with continuous input types, such as fluctuating background traffic. DQL enhances traditional Q-learning by integrating it with deep neural networks (DNNs), enabling the efficient processing of continuous observations and complex, high-dimensional state spaces. This capability is crucial for developing strategies that can adapt to and effectively manage the dynamics of background traffic. In addition to our main DQL approach, we have also implemented tabular Q-learning as a baseline method to provide a comparative framework. This allows us to assess the enhancements offered by deep learning approaches over more traditional reinforcement learning techniques.

Furthermore, to explore a broader range of neural network-based methods and enhance our system’s adaptability and performance, we have implemented variations of deep Q-learning, including Deep Q-Networks (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). These methods collectively cover a wide spectrum of RL strategies, offering effective solutions to exploit network vulnerabilities across various scenarios and traffic conditions.

3.3 System Optimization

In the context of our RL-based system, optimization strategies focus on refining the internal mechanics of the model, such as state sequence length and the scheduling of reward resets. These adjustments are key to maximizing the system’s operational efficiency and the effectiveness of the RL agent in detecting and exploiting network vulnerabilities.

Optimization of State Sequence Length. The length of the state sequence significantly impacts the RL agent’s ability to learn and make decisions. A longer sequence provides more historical context but at the cost of increased computational overhead and potential delays in decision-making. Conversely, a shorter sequence may lead to quicker responses but can result in under-informed decisions due to a lack of sufficient context. Balancing the length of the state sequence is essential for optimizing the trade-offs between decision quality and system performance.

Reward Reset Intervals. Determining how frequently to reset the reward mechanism is vital for maintaining the agent’s motivation to learn and adapt. Too frequent resets may prevent the agent from recognizing long-term consequences of its actions, whereas infrequent resets might cause the agent to persist with outdated strategies that are no longer effective. Optimizing the reward reset intervals helps in keeping the learning process aligned with the dynamic nature of network environments,

ensuring that the agent remains responsive to new challenges and opportunities for optimization.

These optimizations contribute to a robust, agile, and scalable system capable of meeting the demands of modern network environments.

Chapter 4

Implementation

In this section, we introduce the implementation of the **SketchProbe** system, focusing on the networking simulator and the RL agent implementation.

4.1 Network Simulator

The implementation of the **SketchProbe** system utilizes the Parasol networking simulator [16] to create a realistic and controlled environment for evaluating the performance of the attack strategy. Parasol is a framework designed to synthesize data-plane programs, overcoming the limitations of earlier systems by providing a more flexible and expressive approach to network simulation and optimization. The key capabilities of Parasol that make it suitable for simulating and potentially attacking CMS work include:

- (1) **Configurable Program Parameters:** Parasol enables users to modify various aspects of a data-plane program’s implementation. For CMS, parameters like hash functions, the number of hash tables, and the size of each table could be adjusted. This flexibility allows the simulation to explore how changes in these parameters impact the performance of CMS.

- (2) **Simulation with Realistic Network Conditions:** Parasol uses traffic traces from specific network environments to optimize the data-plane program. This feature is crucial for testing CMS under different types of network behaviors, including adversarial conditions. By simulating CMS with traffic that might include deliberate attempts to exploit its vulnerabilities (e.g., flooding one of the hash buckets to cause frequent collisions), Parasol can assess the resilience of CMS against such attacks.
- (3) **Comprehensive Simulation Tools:** The sketching language and the Python-based objective computation in Parasol allow for detailed and nuanced definitions of performance metrics and simulation behaviors. This capability ensures that the testing and optimization of CMS can include a wide range of possible network scenarios and attack vectors, making the simulation outcomes highly relevant and practically useful for enhancing the security and robustness of CMS implementations in real-world networks.

Parasol allows the generation of diverse background traffic patterns and the simulation of various network conditions consisting of a sketching language, making it suitable for testing the adaptability and effectiveness of **SketchProbe**.

4.2 RL Agent

We implemented the RL agent in **SketchProbe** using several learning methods. Our primary framework for these implementations utilizes the OpenAI Gym [2], which provides a versatile and standardized environment for developing and testing reinforcement learning algorithms. This platform facilitates the simulation of various network scenarios, allowing our agent to interact dynamically with the environment and adapt its strategies accordingly.

Tabular RL. As a comparative baseline, we implement Tabular RL, which is ideal for scenarios with discrete and not excessively large state or action spaces. We configure the approach to handle a scenario characterized by a discrete action space, where the agent can choose from 501 possible actions, ranging from sending 0 to 500 packets. In Tabular RL, we maintain a Q-table with each state corresponding to a distinct network condition and each column representing one of the 501 possible packet-sending actions. The values in this Q-table represent the expected future rewards for taking a given action from a particular state. We set the learning rate, α , to 0.1. This value is chosen to provide a balance between learning speed and stability, allowing the agent to update its Q-values effectively without oscillating or diverging. The discount factor, γ , is set at 0.99, this value reflects the importance of future rewards, encouraging the agent to consider long-term outcomes while still giving weight to immediate gains. This is particularly important in a network environment where actions can have extended consequences.

Deep Q-Network (DQN). The DQN serves as our foundational learning model, utilizing a DNN to approximate the Q-values associated with different actions given certain states. This method is particularly suited for environments with high-dimensional state spaces, such as those involving detailed network traffic patterns and sketch statistics. Our DQN consists of an input layer that captures the state, one hidden layer that learns to interpret these inputs, and an output layer that predicts Q-values for each possible action. We employ an epsilon-greedy strategy, where the agent initially explores the environment by choosing actions randomly but gradually shifts to exploiting known rewarding actions as it learns. The exploitation-exploration rate, ϵ , is initially set to 0.9 and decayed over time to 0.05, ensuring a balance between exploring new strategies and exploiting learned behaviors. To further stabilize the learning process and reduce correlation between consecutive updates, DQN utilizes a

replay buffer. This buffer stores previous experiences, which are sampled randomly to perform learning updates.

Actor-Critic Methods (A2C and PPO). Both A2C and PPO refine the policy gradient approach by decoupling the policy decision-making (actor) from the value estimation (critic), enhancing stability and learning efficiency. Actor net outputs a probability distribution over actions, guiding which actions to take based on the policy. In the context of the Actor-Critic and PPO methods, the network configurations and parameters are crucial for the effective learning and decision-making of the RL agent. We set the discount factor, γ , to 0.99, learning rate, α , to 0.001. Target update is set to every 10 episodes.

Chapter 5

Evaluation

In this section, we describe the methodology for experimentally training and evaluating `SketchProbe` with the realistic case where there is background traffic. Our experiments cover a broad set of RL-based attack methods. Our results demonstrate that:

- (1) Our `SketchProbe` attack creates higher error rate than baseline cases, which include no attacker case and random attack case.
- (2) Our `SketchProbe` attack can be implemented in different RL-based methods, and they achieve different levels of performance.
- (3) Our `SketchProbe` attack works with different traffic distributions, and the distribution affects the vulnerability of the sketch.

5.1 Methodology

Simulation Configuration. The Parasol simulator is configured to replicate a wide range of network conditions through adjustable parameters that dictate the scale and behavior of network traffic [16]. This configurability is critical for ensuring that `SketchProbe` can be evaluated against both expected and adverse network conditions.

The simulation environment and traffic generation are defined by specific network parameters encoded in the system’s configuration files. Our sketch has a fixed size of 2 rows and 128 columns. We also set various number of packets and number of flows to control the total number of packets and flows in the simulation allowing for scalability testing under high-load conditions. Distribution can be set to different levels of skewness, ranging from very skewed based on Zipfian distribution to randomly generated packets. The Zipfian distribution is chosen for its ability to model scenarios where some items (e.g., video content, web pages) are vastly more popular than others, which is a common occurrence in networks. In addition, rate of sending packets and whether or not to send background traffic are parameters that determine the pacing of the traffic and the inclusion of background traffic, respectively, essential for testing **SketchProbe** in diverse networks and ensuring it can differentiate between attack and normal traffic patterns. Background traffic further refines the simulation by specifying the amount of non-attack traffic that mimics typical user behavior on the network.

Metrics. In the context of evaluating the RL agent in **SketchProbe**, we mainly evaluate and compare the error rate caused by collisions generated by the agent for our methods. The error rate measures the discrepancy between the true counts of network flows and the counts estimated by the CMS due to the agent’s actions. It is a direct indicator of the effectiveness of the attack.

Training. We train every RL implementation for 200 rounds, and compare the training curves of different learning methods. Figure 5.1 illustrates the testing error rates of four different reinforcement learning methods across numerous training rounds. The x-axis enumerates the training rounds, while the y-axis quantifies the error rate percentage observed during testing. Each line represents one of the RL methods: Tabular RL, Basic DQN, A2C, and PPO. As training progresses, the Tabular RL method shows a gradual improvement, stabilizing at a lower error rate. Basic DQN and A2C exhibit a more pronounced increase in error rate before reaching their re-

spective convergence points, marked by circles. PPO, however, demonstrates a swift escalation in error rate, reaching a plateau that signifies its convergence point, beyond which no significant learning improvement is observed. This convergence point is critical, and PPO achieves the earliest convergence point, indicating achieving best performance the fastest out of the methods.

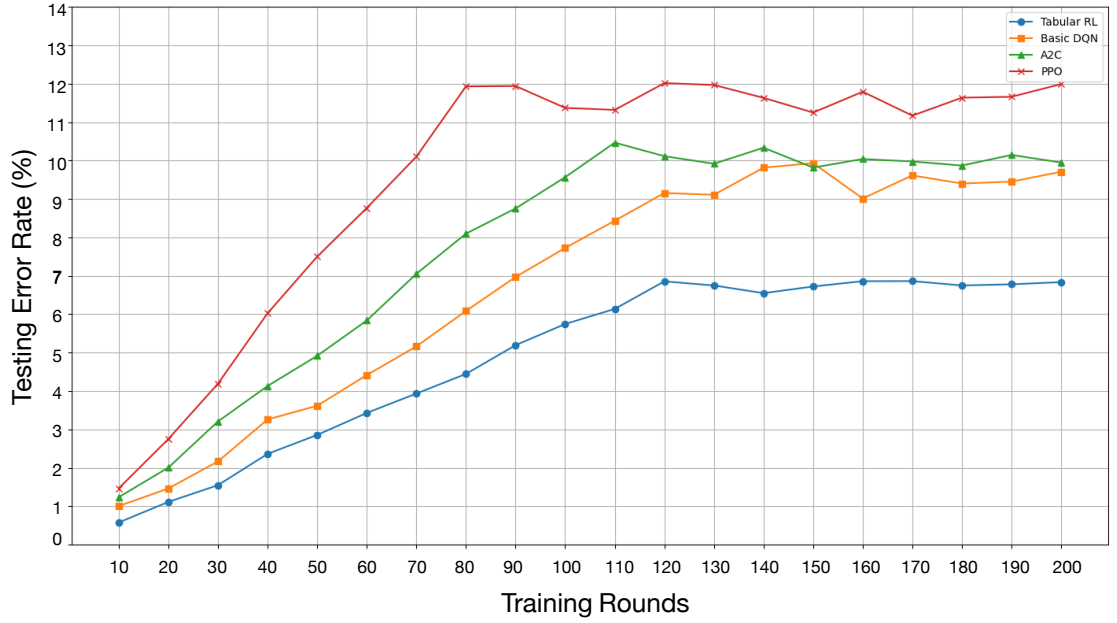


Figure 5.1: Different learning-based attacks training curves. The graph shows error rate across training rounds for different RL methods.

Baselines. The baselines employed in the evaluation of **SketchProbe** are as follows:

- (1) No attack scenario: In the no attack baseline, there are no attacking traffic flow, and the error are caused solely by collision of the background traffic itself, which should be bounded and low.
- (2) Random attack scenario: For the random attack baseline, each iteration involves randomly selecting packets for attack on the same background traffic flow, and the process is repeated across ten different background traffic flows.

These baselines serve as points of comparison to evaluate the performance of **SketchProbe**. The evaluation involves training **SketchProbe** and comparing its performance to these baselines based on error rate caused by collisions.

5.2 Results

To evaluate **SketchProbe**, we first compare **SketchProbe** with the baseline cases that does not have an attacker and that has an attacker attacking randomly.

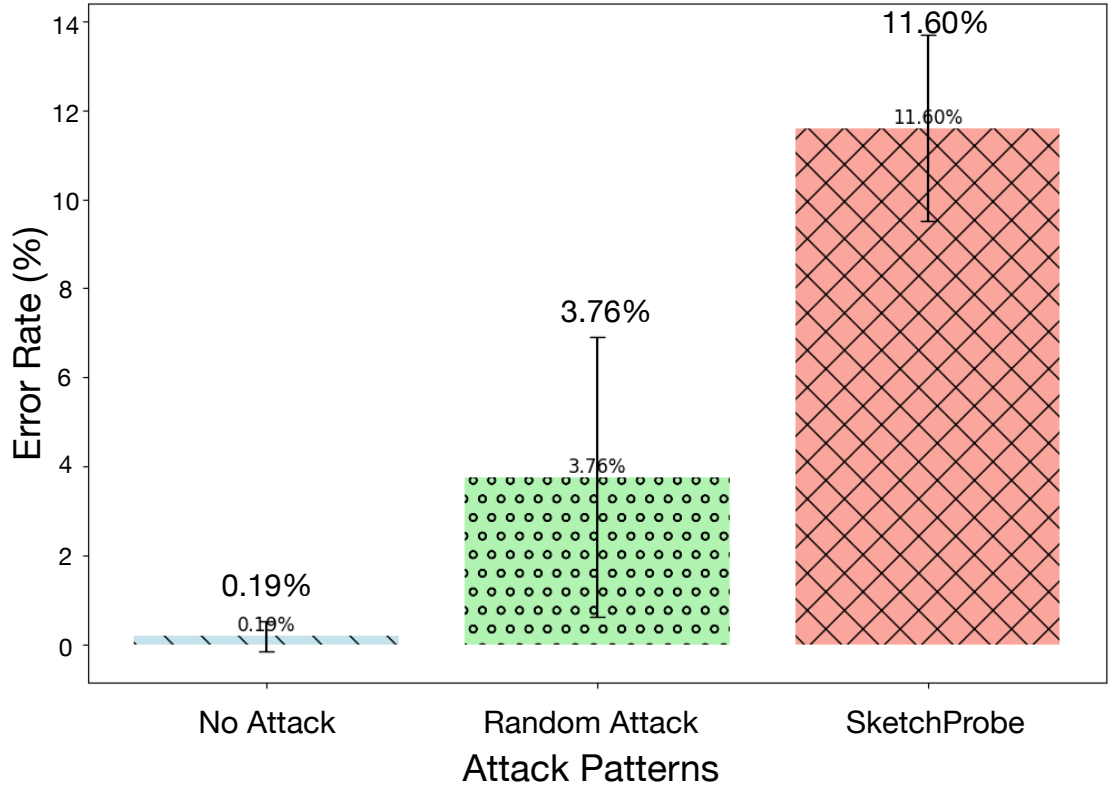


Figure 5.2: Compare **SketchProbe** with no attack and random attack baselines. This graph shows error rate comparison by attack patterns.

Figure 5.2 presents a comparative analysis of error rates arising from different attack patterns on a network monitoring system employing sketch-based data structures. The error rate, represented as a percentage, serves as a metric for the accuracy

of the sketch in tracking network traffic. Three distinct scenarios are considered: no attack, random attack, and our targeted RL-based attack method, **SketchProbe**. In the absence of any attack, the observed error rate is minimal, averaging at 0.19%, which can be attributed to collisions within background traffic — caused by different data streams inadvertently mapping to the same counters in the sketch data structure. This scenario serves as a baseline representing the system’s inherent noise level without adversarial interference. The random attack scenario, wherein a random number of packets between 0 and 500 are sent in each window, results in a moderate increase in the error rate to 3.76%. This increment indicates a higher collision rate due to the added traffic, which, while random, still introduces additional strain on the system’s ability to maintain accurate statistics. Our proposed attack method, demonstrated by **SketchProbe**, exhibits a significant spike in the error rate, culminating at 11.60%. This substantial rise indicates the effectiveness of the RL-based strategy in identifying and exploiting vulnerabilities within the sketch-based monitoring system. The targeted nature of the attack, as opposed to random packet injections, leads to a more pronounced disruption of the system’s accuracy. Each pattern’s experiment is repeated across ten different background traffic flow runs.

Next, we evaluate different implementations of RL attack methods as represented in Figure 5.3. This depicts an error rate comparison across various reinforcement learning methods in attacking the sketch-based application. The methods compared include Tabular RL, Basic Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). We hypothesize that studying methods which incorporate more complex state information and learning strategies, such as PPO, will be more effective at exploiting the CMS’s vulnerabilities compared to simpler methods like Tabular RL, because sophisticated RL algorithms can better understand and manipulate the application state to induce maximum error.

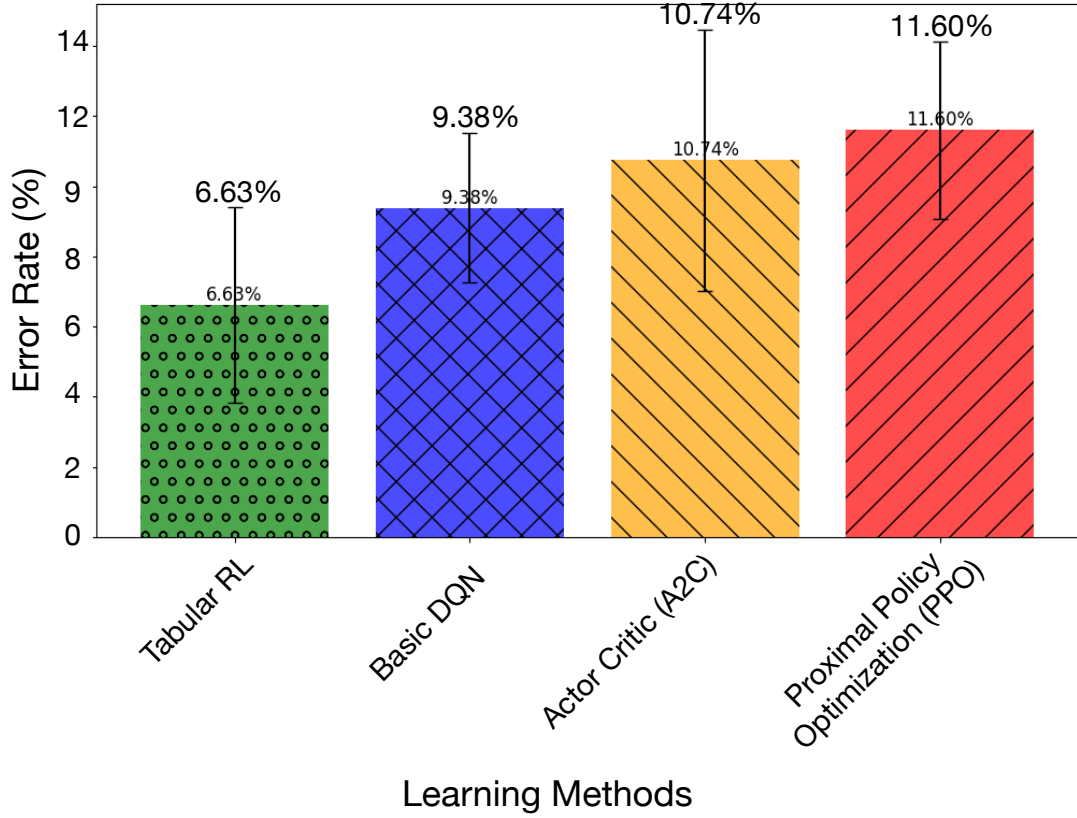


Figure 5.3: Compare different learning-based attack implementations. This graph shows error rate comparison by learning methods.

Tabular RL results in a lower error rate of 6.63%, indicative of its more basic nature and potentially limited capability in complex environments. DQN exhibits a higher error rate of 9.38%; the DQN method leverages a neural network to approximate the Q-function, allowing for a more nuanced attack strategy than the tabular approach. A2C has an error rate of 10.74%; the A2C method improves upon DQN by utilizing separate networks for policy (actor) and value (critic) estimation, resulting in a more effective learning and attack execution process. At the highest end, PPO achieves an error rate of 11.60%, signifying the most effective learning strategy among those tested. PPO's sophisticated policy optimization technique, which strikes a balance between exploration and exploitation, accounts for this high error induction capability. This figure illustrates the varying effectiveness of different RL methods in

compromising the accuracy of network monitoring systems. It highlights the potential for sophisticated RL techniques like PPO to significantly disrupt network traffic monitoring performance. In addition, we also record the overhead for these different learning-based methods in Table 5.1.

Method	Train Time/Ep (sec)	Max Packets/Ep	Ep to Converge
Tabular RL	3.028	500	120
Basic DQN	5.063	475	100 – 150 (depends on rate)
Actor Critic	5.021	375	100
PPO	5.026	457	80

Table 5.1: Training performance and convergence of RL methods.

Finally, we evaluate different traffic flow distributions. This experiment is designed to assess the robustness of the **SketchProbe** under different traffic flow distributions since traffic distributions are fundamental characteristics of network data that can affect the probability of hash collisions within CMS. We aim to simulate realistic and challenging network scenarios from moderate to highly skewed distributions. Figure 5.4 presents a comparative analysis of error rates associated with different traffic distribution types affecting the performance. The graph delineates two categories of distribution: Random and Skewed, where we use the Zipfian distribution generator from numpy [15] to generate a set of parameter values 1.3, 1.8, and 2 to describe skewness, where the parameter value controls the degree of skewness. A higher parameter value represent a more skewed distribution, the more skewed the distribution is (i.e. the smaller number of flows the packet trace will have).

The random distribution exhibits a relative high error rate at 10.86%, but with a more fluctuated results across different runs. The skewed distribution with 1.3 as parameter has the highest error rate of 11.60%, with a more stable results compared with the random distribution. In addition, the more skewed the traffic is, the harder it is to successfully attack the traffic since more skewed distribution means there is a smaller number of flows with given the same number of packets. This results in

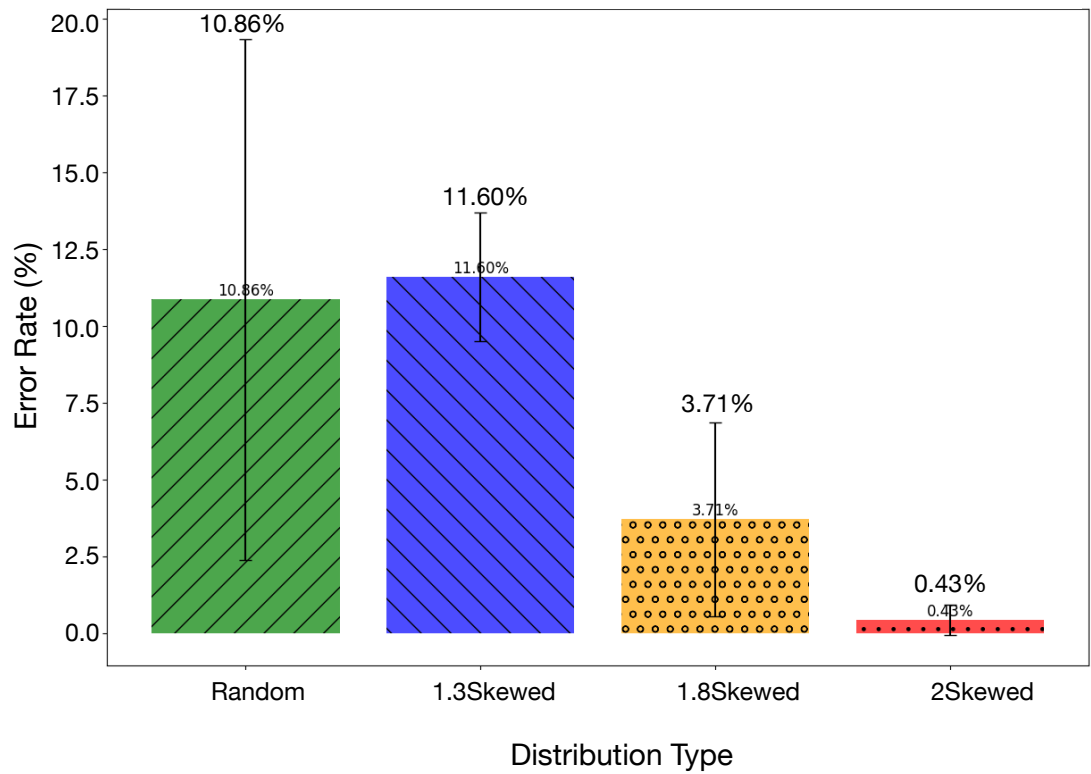


Figure 5.4: Compare different attack traffic distribution. This graph shows error rate comparison by distribution types.

fewer hash collisions. When the traffic is extremely skewed with a parameter of 2, it demonstrates the minimal impact with an error rate of only 0.43%.

Chapter 6

Related Work

6.1 Sketching Data Structures

In networking, a sketch is a probabilistic data structure that allows for the efficient processing of large amounts of data in real time. Sketches are used to summarize large datasets, extract features, and perform approximate computations on data streams in a scalable and memory-efficient way. There are many existing sketches that serve different functionalities. CMS [8] works by using a set of hash functions to map elements to different positions in a fixed-size array, and storing the minimum count of each element across all the hash functions. Bloom Filter [1] works by hashing an element to a set of positions in a bit array. Elastic Sketch [29] is adaptive to current traffic characteristics. However, the process of compressing and merging these sketches, as detailed in the original paper, involves multiple steps and computational overheads that may not be efficiently executable on programmable switches. These switches typically operate under strict performance and resource constraints, and the computational intensity required by Elastic Sketch could potentially hinder its practical deployment in such environments. However, the process of compressing and

merging these sketches involves multiple steps and computational overheads that may not be efficiently executable on programmable switches.

Rotating Networking Data Structures. The idea of rotating networking data structures is introduced in ConQuest [6], a novel approach for fine-grained queue measurement in the data plane. ConQuest utilizes the concept of rotating snapshots to efficiently analyze a FIFO queue with bounded length served by a constant-rate link. The key idea is to maintain a constant number of snapshots, each covering a time window of length T , where T is chosen based on the total number of snapshots (h) needed to approximately cover the entire queue. Each snapshot in ConQuest has a specific role, either reading, writing, or cleaning, at any given time, and the roles of the snapshots rotate in a round-robin fashion synchronized with the progression of the time windows. Every T seconds, which corresponds to the chosen snapshot time window length, the writing snapshot becomes a reading snapshot, a reading snapshot shifts to cleaning, and the cleaning snapshot moves to writing. This cycle ensures continuous, real-time data processing without downtime. This systematic rotation and role assignment allow ConQuest to efficiently manage memory and processing power, adapting dynamically to the flow of traffic without requiring intensive control plane intervention for snapshot management. The process is highly optimized for environments like programmable switches where processing speed and memory efficiency are critical.

6.2 RL in Networking

In recent years, RL has gained significant attention and demonstrated remarkable potential in enhancing various aspects of network management and optimization. A prominent example of RL’s efficacy in networking can be observed in the context of Pensieve [21]. Pensieve is a system that enhances adaptive bitrate (ABR) algorithms

for client-side video players by utilizing RL. It dynamically learns to select optimal bitrates based on real-time observations from video players. This approach allows it to adapt to varying network conditions and QoE objectives without relying on predefined models. Pensieve and SketchProbe both utilize reinforcement learning but are applied in different contexts and for distinct purposes. **SketchProbe** targets the robustness of network monitoring tools that use sketches.

Gilad *et al.* discusses a novel framework designed to assess and enhance the robustness of network protocols using RL [12]. The paper addresses the performance issues in network protocols such as routing, congestion control, and video streaming using RL. The primary aim is to enhance the robustness of these protocols by identifying conditions under which they fail. This involves generating adversarial network conditions that challenge the protocols beyond their typical operational boundaries. It leverages RL to simulate network environments that can potentially cause protocol failures, thus allowing for an assessment of protocol robustness and guiding necessary adjustments to improve their resilience. While both this paper and **SketchProbe** leverage RL to test systems under adverse conditions, this paper is focused on protocols affecting direct network operations, whereas **SketchProbe** is concerned with the backend tools that monitor and analyze network traffic.

Chapter 7

Concluding Remarks

7.1 Conclusion

In this work, we presented **SketchProbe**, a system that utilizes Reinforcement Learning (RL) to identify adversarial traffic patterns affecting sketch-based applications. Unlike traditional approaches that rely on manual testing and predefined heuristics, **SketchProbe** dynamically adapts to the evolving landscape of network threats through continuous learning from network behavior. By automating the identification process, **SketchProbe** not only reduces the time and effort required to detect potential vulnerabilities but also enhances the depth and breadth of security testing. Our evaluations demonstrate that **SketchProbe** significantly improves the ability to pinpoint critical scenarios that could severely impact the functionality and reliability of sketch-based applications. This capability is particularly crucial in maintaining the integrity of network monitoring in environments prone to sophisticated and irregular traffic patterns.

7.2 Future Work

The exploration of reinforcement learning in **SketchProbe** has opened new avenues for enhancing and adapting network security mechanisms. While our current focus has been predominantly on managing workload or volume of traffic using approximate data structures like Count-Min Sketch, future work can extend and diversify the range of adaptive strategies and the scope of our analysis in two significant ways:

- (1) **Generalization to Various Sketch Designs:** Sketch sizes in network monitoring tools are often dictated by the available memory and the expected volume of traffic. Larger sketches can handle higher traffic volumes without significant loss of accuracy, while smaller sketches may be used in resource-constrained environments. To generalize **SketchProbe** across various sketch sizes, adaptive learning algorithms that can adjust their parameters based on the sketch size could be developed. For example, modifying the learning rate or the size of the neural network layers based on the sketch dimensions may help maintain performance regardless of sketch size.
- (2) **Generalization to Other Approximate Data Structures:** The current methodologies developed in **SketchProbe** are tailored to the specifics of CMS-based systems. However, the principles and techniques could be generalized to other approximate data structures used in network environments, such as Bloom filters, HyperLogLog, or Quotient filters. Each of these structures offers unique benefits and challenges in handling network traffic data, particularly in the context of estimating set membership or frequency of elements.
- (3) **Diversification of Attack Strategies:** Currently, the RL-based strategies in **SketchProbe** are designed to manipulate traffic volume to test the robustness of network systems. Looking ahead, there is potential to diversify attack vectors to include manipulations of packet header fields, which can affect how traffic

is classified and managed by network protocols. This approach could reveal vulnerabilities related to protocol-specific behaviors or to the misclassification of traffic, which are critical in maintaining network security and efficiency. Another promising direction is to vary the timing and behavioral patterns of the attacks more complexly. By incorporating strategies that adapt not just the volume but also the timing of traffic bursts, attacks can become more sophisticated and harder to detect. This includes tactics like mimicking legitimate user behaviors or strategically timing attacks to coincide with peak network usage periods, thereby challenging existing security measures more effectively.

These future directions not only broaden the scope of **SketchProbe** but also contribute to a deeper understanding of network dynamics and security. This proactive and expansive approach will be crucial in developing more robust and adaptive network security technologies.

Bibliography

- [1] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- [3] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.
- [4] Andrew T Campbell, Herman G De Meer, Michael E Kounavis, Kazuho Miki, John B Vicente, and Daniel Villela. A survey of programmable networks. *ACM SIGCOMM Computer Communication Review*, 29(2):7–23, 1999.
- [5] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.
- [6] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 15–29, New York, NY, USA, 2019. Association for Computing Machinery.
- [7] David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic data structures in adversarial environments. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1317–1334, 2019.
- [8] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [9] Guillaume Dewaele, Kensuke Fukuda, Pierre Borgnat, Patrice Abry, and Kenjiro Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the 2007 workshop on Large scale attack defense*, pages 145–152, 2007.
- [10] Guhao Feng and Han Zhong. Rethinking model-based, policy-based, and value-based reinforcement learning via the lens of representation complexity, 2023.

- [11] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.
- [12] Tomer Gilad, Nathan H. Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. Robustifying network protocols with adversarial examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 85–92, New York, NY, USA, 2019. Association for Computing Machinery.
- [13] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.
- [14] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.
- [15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [16] Mary Hogan, Devon Loehr, John Sonchack, Shir Landau Feibish, Jennifer Rexford, and David Walker. Automated optimization of parameterized data-plane programs with parasol, 2024.
- [17] Tomasz Bartkowiak (<https://stats.stackexchange.com/users/230011/tomasz-bartkowiak>). What is the difference between policy-based, on-policy, value-based, off-policy, model-free and model-based? Cross Validated. URL:<https://stats.stackexchange.com/q/466950> (version: 2021-05-20).
- [18] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. Toward nearly-zero-error sketching via compressive sensing. In *NSDI*, pages 1027–1044, 2021.
- [19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [20] Elie F Kfoury, Jorge Crichigno, and Elias Bou-Harb. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE access*, 9:87094–87155, 2021.

- [21] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 197–210, New York, NY, USA, 2017. Association for Computing Machinery.
- [22] Yossi Matias, Jeff Vitter, and Neal Young. Approximate data structures with applications, 2002.
- [23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
- [24] Mark EJ Newman. Power laws, pareto distributions and zipf’s law. *Contemporary physics*, 46(5):323–351, 2005.
- [25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [26] Benedikt Sigurleifsson, Aravindan Anbarasu, and Karl Kangur. An overview of count-min sketch and its applications. 2019.
- [27] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
- [28] Frank Swiderski and Window Snyder. *Threat modeling*. Microsoft Press, 2004.
- [29] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.