FINE-GRAINED, CONTENT-AGNOSTIC NETWORK TRAFFIC ANALYSIS FOR MALICIOUS ACTIVITY DETECTION

by

YEBO FENG

A DISSERTATION

Presented to the Department of Computer Science
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 2023

DISSERTATION APPROVAL PAGE

Student: Yebo Feng

Title: Fine-grained, Content-agnostic Network Traffic Analysis for Malicious Activity Detection

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer Science by:

Original approval signatures are on file with the University of Oregon Division of Graduate Studies.

Degree awarded June 2023

DISSERTATION ABSTRACT

Yebo Feng

Doctor of Philosophy

Department of Computer Science

June 2023

Title: Fine-grained, Content-agnostic Network Traffic Analysis for Malicious Activity Detection

The rapid evolution of malicious activities in network environments necessitates the development of more effective and efficient detection and mitigation techniques. Traditional traffic analysis (TA) approaches have demonstrated limited efficacy and performance in detecting various malicious activities, resulting in a pressing need for more advanced solutions. To fill the gap, this dissertation proposes several new fine-grained network traffic analysis (FGTA) approaches. These approaches focus on (1) detecting previously hard-to-detect malicious activities by deducing fine-grained, detailed application-layer information in privacy-preserving manners, (2) enhancing usability by providing more explainable results and better adaptability to different network environments, and (3) combining network traffic data with endpoint information to provide users with more comprehensive and accurate protections.

We begin by conducting a comprehensive survey of existing FGTA approaches. We then propose CJ-Sniffer, a privacy-aware cryptojacking detection system that efficiently detects cryptojacking traffic. CJ-Sniffer is the first approach to distinguishing cryptojacking traffic from user-initiated cryptocurrency mining traffic, allowing for fine-grained traffic discrimination. This level of fine-grained traffic discrimination has proven challenging to accomplish through traditional TA methodologies. Next, we introduce BotFlowMon, a learning-based, content-agnostic approach for detecting online social network (OSN) bot traffic, which has posed a significant challenge for detection using traditional TA strategies. BotFlowMon is an FGTA approach that relies only on content-agnostic flow-level data as input and utilizes novel algorithms and techniques to classify social bot traffic from real OSN user traffic. To enhance the usability of FGTA-based attack detection, we propose a learning-based DDoS detection approach that emphasizes both explainability and adaptability. This approach provides network administrators with insightful explanatory information and adaptable models for new network environments. Finally, we present a reinforcement learning-based defense approach against L7 DDoS attacks, which combines network traffic data with endpoint information to operate. The proposed approach actively monitors and analyzes the victim server and applies different strategies under different conditions to protect the server while minimizing collateral damage to legitimate requests.

Our evaluation results demonstrate that the proposed approaches achieve high accuracy and efficiency in detecting and mitigating various malicious activities, while maintaining privacy-preserving features, providing explainable and adaptable results, or providing comprehensive application-layer situational awareness. This dissertation significantly advances the fields of FGTA and malicious activity detection.

This dissertation includes published and unpublished co-authored materials.

CURRICULUM VITAE

NAME OF AUTHOR: Yebo Feng

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
Yangzhou University, Yangzhou, Jiangsu, China

DEGREES AWARDED:

Doctor of Philosophy, Computer Science, 2023, University of Oregon
Master of Science, Computer Science, 2018, University of Oregon
Bachelor of Engineering, Computer Science and Technology, 2016, Yangzhou University

AREAS OF SPECIAL INTEREST:

Network Security & Privacy: intrusion detection; DDoS defense; social network security & Privacy.
Network Traffic Analysis: content-agnostic network traffic analysis; large-scale traffic measurement.
Blockchain & Web 3.0 Security: blockchain traffic analysis; DeFi security & privacy, DEX & AMM security.
Machine Learning in Security: usable ML-based attack detection and defense system.

PROFESSIONAL EXPERIENCE:

Research Assistant, University of Oregon, 2023
Teaching Assistant (CS 451 Database Processing), University of Oregon, 2022
Visiting Research Assistant, University of Southern California, 2022
Researcher, Ripple, 2022
Research Assistant, University of Oregon, 2021-2022
Research Associate, Ripple, 2021
Research Assistant, University of Oregon, 2018-2021

Research Intern, China Academy for Information and Communications Technology (CAICT), 2020
Teaching Assistant (CIS 451 Database Processing, CIS 110 Fluency with Information Technology, and CIT 281 Web Applications Development I), University of Oregon, 2017-2018
Research Assistant, University of Oregon, 2017
Software Engineer, Gaobiao Xianjin Co, Ltd., 2015-2016
Software Engineer Intern, Chinasoft International Ltd., 2015

GRANTS, AWARDS AND HONORS:

RAID Student Travel Grant, Organizing Committee of RAID, 2022
Ripple Graduate Fellowship (2021 - 2022), Ripple, 2021
Ripple Graduate Fellowship (2020 - 2021), Ripple, 2020
KDD Student Registration Award, National Science Foundation, 2020
Ripple Graduate Fellowship (2019 - 2020), Ripple, 2019
Gurdeep Pall Graduate Student Fellowship, University of Oregon, 2019
IEEE CNS Best Paper Award, IEEE ComSoc, 2019
IEEE CNS Student Travel Grant, IEEE ComSoc, 2019
Graduate Teaching Fellowship, University of Oregon, 2017
Second-Class Chancellor's Scholarship, Yangzhou University, 2015

PROFESSIONAL SERVICES:

Reviewer, IEEE Transactions on Dependable and Secure Computing (TDSC)
Reviewer, IEEE Transactions on Information Forensics & Security (TIFS)
Reviewer, IEEE Transactions on Mobile Computing (TMC)
Reviewer, IEEE Journal on Selected Areas in Communications (JSAC)
Reviewer, ACM Transactions on Knowledge Discovery from Data (TKDD)
Reviewer, IEEE Communications Surveys & Tutorials (COMST)
Reviewer, IEEE Internet of Things Journal (IoTJ)
Reviewer, Security and Communication Networks (SCN)
Reviewer, Financial Innovation (FIN)
Reviewer, IEEE/ACM International Symposium on Quality of Service (IWQoS), 2021-2022
Reviewer, ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2023

Reviewer, IEEE International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), 2023

Judging Committee (Wave 3-5), XRPL Grants, 2022-2023

Program Committee, International Cryptoasset Analytics Workshop (CAAW): 2023

Program Committee, ACM International Conference on Information and Knowledge Management (CIKM), 2022

Technical Program Committee, International Conference on Cyber-Technologies and Cyber-Systems (CYBER), 2020-2022

Technical Program Committee, International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 2020-2022

International Program Committee, Blockchain and Cryptocurrency Congress (B2C), 2022

Artifact Evaluation Committee, USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022

Artifact Evaluation Committee, USENIX Annual Technical Conference (ATC), 2022

Web Team Member, ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2021

Student Volunteer, ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2020

PUBLICATIONS:

Yebo Feng, Jun Li, and Thanh Nguyen. Towards Intelligent Application-layer DDoS defense with reinforcement learning. 2023. **In Preparation**.

Jun Li, Devkishen Sisodia, Yebo Feng, Lumin Shi, Mingwei Zhang, Samuel Mergendahl, Christopher Early, and Peter Reiher. Toward Adaptive Distributed Filtering of DDoS Traffic. 2023. **In Preparation**.

Yebo Feng, Jun Li, Devkishen Sisodia, and Peter Reiher. On Explainable and Adaptable Detection of Distributed Denial-of-Service Traffic. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023. **In Submission**.

Yebo Feng, Jun Li, and Jelena Mirkovic. Unmasking the Internet: A Survey of Fine-Grained Network Traffic Analysis. *IEEE Communications Surveys & Tutorials (COMST)*, 2023. **In Submission**.

Jiahua Xu, Simon Cousaert, Krzysztof Paruch, and Yebo Feng. SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols. *ACM Computing Surveys (CSUR)*, 2023.

Jiahua Xu and Yebo Feng. Reap the Harvest on Blockchain: A Survey of Yield Farming Protocols. *IEEE Transactions on Network and Service Management (TNSM)*, 2023.

Yebo Feng, Jun Li, and Devkishen Sisodia. CJ-Sniffer: Measurement and Content-Agnostic Detection of Cryptojacking Traffic. *The 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022.

Yebo Feng, Jiahua Xu, and Lauren Weymouth. University Blockchain Research Initiative (UBRI): Boosting blockchain education and research. *IEEE Potentials*, 2022.

Teng Li, Siwei Yin, Runze Yu, Yebo Feng, Lei Jiao, Yulong Shen, and Jianfeng Ma. CoAvoid: Secure, Privacy-Preserved Tracing of Contacts for Infectious Diseases. *IEEE Journal on Selected Areas in Communications (JSAC)*, 2022.

Yebo Feng. Toward Finer Granularity Analysis of Network Traffic. *University of Oregon, Area Exam Report*, 2022.

Jelena Mirkovic, Yebo Feng, and Jun Li. Measuring Changes in Regional Network Traffic Due to COVID-19 Stay-at-Home Measures. *arXiv preprint*, 2022.

Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. Towards Learning-Based, Content-Agnostic Detection of Social Bot Traffic. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2021.

Yebo Feng, Devkishen Sisodia, and Jun Li. POSTER: Content-Agnostic Identification of Cryptojacking in Network Traffic. *The 15th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, 2020.

Yebo Feng and Jun Li. Toward Explainable and Adaptable Detection and Classification of Distributed Denial-of-Service Attacks. *The First KDD Workshop on Deployable Machine Learning for Security Defense (MLHat)*, 2020.

Yebo Feng, Jun Li, and Thanh Nguyen. Application-Layer DDoS Defense with Reinforcement Learning. *IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020.

Yebo Feng and Jun Li. Towards Explicable and Adaptive DDoS Traffic Classification. *The 21st Passive and Active Measurement Conference (PAM) - Poster*, 2020.

Yebo Feng. Towards Intelligent Defense against Application-Layer DDoS with Reinforcement Learning. *University of Oregon, Directed Research Project (DRP) Report*, 2019.

Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. BotFlowMon: Learning-Based, Content-Agnostic Identification of Social Bot Traffic Flows. *IEEE 7th Conference on Communications and Network Security (CNS)*, 2019.

Yebo Feng. BotFlowMon: Identify Social Bot Traffic with NetFlow and Machine Learning. *Master Thesis, Scholars' Bank - University of Oregon*, 2018.

ACKNOWLEDGEMENTS

Pursuing a Ph.D. has been a long journey, but I have been fortunate to meet many wonderful teachers, friends, and colleagues along the way. Their support and encouragement have made this journey both enjoyable and inspiring for me.

I am deeply grateful to my advisor, Jun Li, for his unwavering guidance and mentorship throughout my Ph.D. journey. His invaluable lessons on critical thinking, diligent research, and effective writing have significantly contributed to my growth and development.

I would also like to extend my sincere appreciation to my committee members, Yingjiu Li, Lei Jiao, Thanh Nguyen, and Kathryn Mills, for their constructive feedback and suggestions. Their guidance has been crucial in helping me reach the final milestone of my Ph.D.

My heartfelt thanks go to my friends and colleagues for their invaluable support, both emotionally and academically. The camaraderie and collaboration we shared have enriched my experience and contributed to my growth. I would like to extend my gratitude to them by mentioning their initials, in alphabetical order: C.S., D.S., D.Y., G.L., H.B., H.T., H.ZX., L.MZ., L.QH., L.W., L.X., M.CY., S.LM., S.M., T.GY., T.Y., W.K., X.JH., Y.FZ., Y.JM., Y.XY., Z.G., Z.RM., and Z.YH. Your friendship and support have truly made a difference.

Finally, I wish to express my deepest gratitude to my parents for their unwavering love, support, and inspiration throughout my life. Their belief in me and their constant guidance have been my driving force during this academic journey. I am eternally grateful for the strong foundation they have provided, upon which I continue to build my dreams.

To my cherished family,

Despite the vast distance between us, your love and support have been my pillars of strength throughout my long academic journey in the US. This dissertation is dedicated to you. Thank you for your unwavering belief in me.

With all my love,

Yebo Feng

TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

Table Page

Table Page

CHAPTER I

INTRODUCTION

Traffic analysis (TA) [137] is a widely-used technique that captures and utilizes network features from the network or transport layers to infer knowledge regarding the ongoing communications. Nowadays, TA is widely used to monitor network events, measure network usages, detect anomalies, and pinpoint intrusions, thereby safeguarding the internet. With decades of research and developments, the security community has proposed a myriad of TA approaches to detect a variety of malicious activities for endpoints in the network, such as worm detection [247, 246], distributed denial-of-service (DDoS) traffic classification [92], botnet detection [184], etc. Compared with endpoint-based detection approaches that directly leverage the operational information from endpoints to discover malicious activities, TA-based approaches feature the following advantages.

– They can be easily deployed on the gateway of a network to monitor all the end points inside. While endpoint-based malicious activity detection approaches have to be deployed on every single node to provide comprehensive protections, which is challenging in most cases.

– They are efficient to operate. As network traffic information is usually coarse-grained summaries of network packet headers, the information to be processed is significantly reduced compared to the operational information on all the nodes. Consequently, with the computing power of a single personal computer, a TA-based malicious activity detection approach can protect a large amount of end points at the line speed.

– They are privacy-preserving as they do not require the sensitive content data of users to detect malicious activities, instead, they leverage content-agnostic network traffic data to infer malicious behaviors. Conversely, endpoint-based malicious activity detection approaches require operational information on the user side to function, such as application-layer logs, content of messages, system operation information, etc.

However, the negative aspects of today's malicious activity detection approaches based on TA are also apparent. Various flaws in traditional TA approaches limit the types of malicious activities that can be detected, the fine-grained visibility of ongoing malicious activities, the accuracy of the detection results, their usability in real-world scenarios, etc. More sophisticated and mature TA approaches are needed to fill the gap.

First of all, the detection coverage of existing approaches is still limited. As network traffic data is usually content-agnostic and only contains partial information about the ongoing communication, it is difficult to deduce many types of fine-grained knowledge about the endpoint behaviors. This is especially true for some application-layer malicious activities, such as online social network (OSN) bots, cryptojacking attacks, and application-layer (L7) DDoS attacks. We need to equip TA approaches with more sophisticated algorithms or techniques to tackle these challenging malicious activities, making the output results fine-grained.

Besides, many TA-based malicious activity detection approaches are weak in usability. For example, many TA approaches are not easily adaptable. Their performance highly depends on the coverage and applicability of the training data. They may work well in one network environment but have significant performance drops in a different network environment. Furthermore, due to the heavy use of machine learning algorithms, many TA-based detection approaches' outputs are inexplicable, just like black boxes. One can hardly fetch explanatory information from the traffic analysis results. In real-world deployments, good explainability is particularly needed by network administrators, as they usually need to manually review and verify the detection results, including eliminating false alarms and avoiding severe collateral damage due to the filtering of traffic from legitimate users. Therefore, we need to propose some improvements to enhance the usability of TA-based detection approaches.

In addition, TA-based malicious activity detection approaches are essentially limited in detecting some application-layer attacks due to the nature of the input data. Thus, finding a promising method to combine TA and endpoint information to provide users with more comprehensive protections is a very significative avenue of future research.

Last but not least, the accuracy of TA-based malicious activity detection approaches require improvements. Generally, the detection accuracy of TA-based approaches is lower than endpoint-based approaches because TA-based approaches only identify malicious activities through the meta data of network traffic, without any knowledge regarding the content or application-layer information. Nonetheless, the detection accuracy, as well as the false positive rate, recall score, and precision score, are vital to the effectiveness of these detection approaches. We thus need to consolidate this aspect by improving existing methodologies.

1.1 Dissertation Statement

As traditional TA approaches have limited efficacy and performance in detecting various malicious activities, we need to develop more sophisticated TA approaches to fill the gaps. **In this dissertation, we propose several new**

fine-grained network traffic analysis (FGTA) approaches for malicious activity detection. These newly proposed approaches are focused on (1) detecting previously hard-to-detect malicious activities by deducing fine-grained, detailed application-layer information in privacy-preserving manners, (2) enhancing the usability by providing more explainable results and better adaptability to different network environments, and (3) combining network traffic data with endpoint information to provide users with more comprehensive and accurate protections.

Specifically, we use the following five projects to achieve our dissertation goals.

1. We first examine the literature that deals with FGTA [148] to investigate the frontier developments in this domain. By comprehensively surveying different approaches toward FGTA, we introduce their input traffic data, elaborate on their operating principles by different use cases, indicate their limitations and countermeasures, and raise several promising future research avenues.

2. To leverage FGTA to broaden the use cases of TA and detect previously hard-to-detect malicious activities, we propose two new FGTA-based detection approaches to tackle two challenging application-layer malicious activities:

    (a) cryptojacking activities, which are unauthorized use of other people's computing resources to mine cryptocurrencies, and

    (b) OSN bots, which are automated accounts that are used to perform malicious activities on OSNs.

3. We demonstrate that the usability of FGTA-based attack detection approaches can be enhanced, as exemplified by our new learning-based DDoS

detection approach that delivers adaptable and explainable DDoS traffic classification.

4. Finally, to address the limited application-layer situational awareness of TA-based methods, as demonstrated in our case study, we propose a blend of FGTA and endpoint-based defense strategies. This combination aims to provide users with more accurate, comprehensive, and tailored protections against L7 DDoS attacks.

1.2 Main Components of the Dissertation

In this section, we briefly introduce the main components of this dissertation. Section 1.2.1 and Section 1.2.2 present two FGTA-based detection approaches to broaden the use cases of TA and detect previously hard-to-detect malicious activities. Section 1.2.3 demonstrates that FGTA can be more usable by proposing a new learning-based DDoS detection approach that features better adaptability and explainability. Section 1.2.4 proposes a blend of FGTA and endpoint-based defense strategies to provide users with more comprehensive and accurate protections against L7 DDoS attacks, demonstrating that we can combine network traffic data with endpoint information to enhance the application-layer situational awareness of FGTA-based approaches.

1.2.1 Measurement and Content-Agnostic Detection of Cryptojacking Traffic. With the continuous appreciation of cryptocurrency, cryptojacking, the act by which computing resources are stolen to mine cryptocurrencies, is becoming more rampant. Therefore, in one chapter of this dissertation, we conduct a measurement study on cryptojacking network traffic and propose CryptoJacking-Sniffer (CJ-Sniffer), an easily deployable, privacy-aware approach based on FGTA to protecting all devices within a network against cryptojacking. Compared with existing approaches that suffer from privacy concerns or high overhead, CJ-Sniffer only needs to access anonymized, content-agnostic metadata of network traffic from the gateway of the network to efficiently detect cryptojacking traffic. In particular, while cryptojacking traffic is also cryptocurrency mining traffic, CJ-Sniffer is the first approach to distinguishing cryptojacking traffic from user-initiated cryptocurrency mining traffic, making it possible to only filter cryptojacking traffic, rather than blindly filtering all cryptocurrency mining traffic as commonly practiced. This fine-grained traffic discrimination is challenging to achieve through traditional TA-based approaches and is also the main contribution of this work.

After constructing a statistical model to identify all the cryptocurrency mining traffic, CJ-Sniffer extracts variation vectors from packet intervals and utilizes a long short-term memory (LSTM) network to further identify cryptojacking traffic. We evaluated CJSniffer with a packet-level cryptomining dataset. Our evaluation results demonstrate that CJ-Sniffer achieves an accuracy of over 99% with reasonable delays.

1.2.2 Learning-based, Content-agnostic Detection of OSN Bot Traffic. Next, we tackle a more challenging problem in the realm of application-layer malicious activity detection—distinguishing OSN bots from real OSN users using FGTA. This project represents our ongoing effort to broaden the range of practical applications for FGTA.

With the fast-growing popularity of OSNs, the security and privacy of OSN ecosystems becomes essential for the public. Among threats OSNs face, malicious social bots have become the most common and detrimental. They are often employed to violate users' privacy, distribute spam, and disturb the financial market, posing a compelling need for effective social bot detection solutions. Unlike traditional social bot detection approaches that have strict requirements on data sources (e.g., private payload information, social relationships, or activity histories), this work proposes a method called BotFlowMon that relies only on content-agnostic flow-level data as input to identify OSN bot traffic. BotFlowMon introduces several new algorithms and techniques to classify social bot traffic from real OSN user traffic, including aggregating network flow records to obtain OSN transaction data, fusing transaction data to extract features and visualize flows, and an innovative density-valley-based clustering algorithm to subdivide each transaction into individual actions.

The evaluation shows BotFlowMon can identify the traffic from social bots with a 96.1% accuracy, which, based on the worst case study on a testing machine, only takes no more than 0.71 seconds on average after it sees the traffic.

1.2.3 Explainable and Adaptable Detection of DDoS Traffic.

Limited usability, stemming from a lack of explainability and adaptability, is another common issue associated with traditional TA-based methods. In this dissertation, we confront this challenge through a dedicated case study. Specifically, we put forward an FGTA-based DDoS detection approach that underlines both explainability and adaptability.

Launched from numerous end-hosts throughout the Internet, a DDoS attack can exhaust the network bandwidth or other resources of a victim, cripple its service, and make it unavailable to legitimate clients. Recently many learning-based approaches attempt to detect DDoS attacks, but their results are often hardly explainable to users and their models are seldom adaptable to new environments. In one chapter of this dissertation, we propose a new learning-based

DDoS detection approach. It detects DDoS attacks via an enhanced k-nearest neighbors (KNN) algorithm, which utilizes a k-dimensional (KD) tree to speed up the detection process, and classifies DDoS sources at a fine granularity according to each IP's risk level. Compared to previous DDoS detection approaches, this approach outputs explanatory information that enables network administrators to easily inspect detection results and make necessary interventions. Moreover, this approach is adaptable in that users do not need to retrain the detection model to have it fit with a new network environment.

We evaluated this approach in both simulated environments and the real world, achieving 95.4% accuracy in detecting DDoS attacks at line speed. In addition, we carried out a human subject study on its explainability, demonstrating that the outputs can help people better understand the attack and make interventions precisely and promptly.

1.2.4 Application-Layer DDoS Defense with Reinforcement Learning. In the end, we illustrate how FGTA can enhance application-layer situational awareness when combined with endpoint-based defense strategies. Specifically, we propose a reinforcement-learning-based approach to L7 DDoS attack defense as a compelling demonstration of this enhanced capability.

L7 DDoS attacks, by exploiting application-layer requests to overwhelm functions or components of victim servers, have become a rising major threat to today's Internet. However, because the traffic from an L7 DDoS attack appears legitimate in transport and network layers, it is difficult for traditional TA-based DDoS solutions to detect and defend against an L7 DDoS attack due to their limited application-layer situational awareness.

We thus propose a new, reinforcement-learning-based approach to L7 DDoS attack defense. It combines FGTA and endpoint-based defense approaches to overcome the weak application-layer situation awareness capability of TA-based approaches. We introduce a multi-objective reward function to guide a reinforcement learning agent to learn the most suitable action in mitigating L7 DDoS attacks. Consequently, while actively monitoring and analyzing the victim server, the agent can apply different strategies under different conditions to protect the victim: When an L7 DDoS attack is overwhelming, the agent will aggressively mitigate as many malicious requests as possible, thereby keeping the victim server functioning (even at the cost of sacrificing a small number of legitimate requests); otherwise, the agent will conservatively mitigate malicious requests instead, with a focus on minimizing collateral damage to legitimate requests.

The evaluation shows that our approach can achieve minimal collateral damage when the L7 DDoS attack is tolerable and mitigate 98.73% of the malicious application messages when the victim is brought to its knees.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter II, we conduct a comprehensive survey of existing FGTA approaches, investigating the frontier developments and gaps in this domain. In Chapter III, we first comperhensively measure the cryptojacking network traffic, and then propose CJ-Sniffer, an easily deployable, privacy-aware, FGTA-based approach to protecting all devices within a network against cryptojacking. In Chapter IV, we leverage FGTA to tackle a more challenging application-layer malicious activity—OSN bot behavior. Specifically, we propose BotFlowMon, a learning-based, content-agnostic approach to identify OSN bot traffic. In Chapter V, we pursue better explainability and adaptability in attack detection by proposing a learning-based DDoS traffic classification approach based on FGTA. As only using network traffic is essentially limited in detecting many application-layer malicious activities, in Chapter VI, we propose reinforcement-learning-based approach to L7 DDoS attack defense by combining both network traffic and endpoint information. In Chapter VII, we forward remaining open issues and indicate several future research avenues. In the end, we conclude this dissertation in Chapter VIII.

1.4 Co-authored Materials

The majority of the content in this dissertation is from published and unpublished research work. Below we connect each chapter to the material and authors that contributed to it.

- Chapter II: Survey of Fine-Grained Network Traffic Analysis
    * Unpublished as Yebo Feng, Jun Li, and Jelena Mirkovic. "Unmasking the Internet: A Survey of Fine-Grained Network Traffic Analysis." *IEEE Communications Surveys and Tutorials*, 2023 [153]. **In submission**.
    * Published as Yebo Feng. "Toward finer granularity analysis of network traffic." *Computer and Information Science, University of Oregon, Technical Report, AREA-202203-Feng*, 2022 [148].
- Chapter III: Measurement and Content-Agnostic Detection of Cryptojacking Traffic
    * Published as Yebo Feng, Jun Li, and Devkishen Sisodia. "CJ-Sniffer: Measurement and Content-Agnostic Detection of Cryptojacking Traffic." *In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), pp. 482-494*, 2022 [155].

    ∗ Published as Yebo Feng, Devkishen Sisodia, and Jun Li. "Poster: Content-agnostic identification of cryptojacking in network traffic." *In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIACCS), pp. 907-909*, 2020 [157].

– Chapter IV: Learning-Based, Content-Agnostic Detection of OSN Bot Traffic

    ∗ Published as Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. "Towards learning-based, content-agnostic detection of social bot traffic." *IEEE Transactions on Dependable and Secure Computing 18, no. 5 (2020): 2149-2163* [152].

    ∗ Published as Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. "BotFlowMon: Learning-based, content-agnostic identification of social bot traffic flows." *In Proceedings of 2019 IEEE Conference on Communications and Network Security (CNS), pp. 169-177*, 2019 [151].

    ∗ Published as Yebo Feng. "BotFlowMon: Identify Social Bot Traffic With NetFlow and Machine Learning." *University of Oregon, Master Thesis*, 2018 [145].

– Chapter V: On Explainable and Adaptable Detection of Distributed Denial-of-Service Traffic

    ∗ Unpublished as Yebo Feng, Jun Li, Devkishen Sisodia, and Peter Reiher. "On Explainable and Adaptable Detection of Distributed Denial-of-Service Traffic." *IEEE Transactions on Dependable and Secure Computing*, 2023. **In submission**.

    ∗ Published as Yebo Feng, and Jun Li. "Toward explainable and adaptable detection and classification of distributed denial-of-service attacks." *In*

Deployable Machine Learning for Security Defense: First International Workshop (MLHat 2020), Proceedings 1, pp. 105-121, 2020 [149].

  ∗ Published as Yebo Feng, and Jun Li. "Towards explicable and adaptive DDoS traffic classification." *In The 21st Passive and Active Measurement Conference as poster*, 2020 [150].

- Chapter VI: Combining Network Traffic and endpoint Knowledge for Intelligent Application-layer DDoS Defense

  ∗ Published as Yebo Feng, Jun Li, and Thanh Nguyen. "Towards Intelligent Application-layer DDoS defense with reinforcement learning." *Transactions on Information Forensics & Security*, 2023. **In preparation**.

  ∗ Published as Yebo Feng, Jun Li, and Thanh Nguyen. "Application-layer DDoS defense with reinforcement learning." *In 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), pp. 1-10*, 2020 [154].

  ∗ Published as Yebo Feng. "Towards Intelligent Defense against Application-Layer DDoS with Reinforcement Learning." *Computer and Information Science, University of Oregon, Technical Report, DRP-201912-Feng*, 2019 [146].

1.5 Acknowledgment

The material presented in this dissertation is partially based upon work supported by the National Science Foundation under Grant No. 1564348, the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00204, and Ripple under the

University Blockchain Research Initiative (UBRI) [158]. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the National Science Foundation, the Department of Homeland Security, or Ripple.

CHAPTER II

SURVEY OF FINE-GRAINED NETWORK TRAFFIC ANALYSIS

Fine-grained traffic analysis (FGTA) is an advance of traffic analysis (TA) that aims to analyze network traffic at a finer granularity for application-layer behavior modeling, fine-grained user activities inferring, or traffic content decoding, only through link-layer or network-layer traffic data, with or without encryptions. Typically FGTA tasks include classifying traffic from different application-layer activities (e.g., Twitter post vs. Tweeter read), different groups of application users (e.g., online social network (OSN) bots vs. normal users), or different user content (e.g., the visiting website).

To systematically study FGTA and further advance the state-of-the-art for detecting malicious activities, it is vital to examine the literature that deals with FGTA, understanding the current gaps and challenges, and identifying the future research directions. In this chapter, we provide a comprehensive survey of the existing FGTA literature, including the state-of-the-art techniques, use cases, countermeasures, and limitations. It not only helps us to understand the background and frontier developments of FGTA, but also provides a solid foundation for our own research.

This chapter is derived in part from the following published and unpublished articles:

- Yebo Feng, Jun Li, and Jelena Mirkovic. "Unmasking the Internet: A Survey of Fine-Grained Network Traffic Analysis." *IEEE Communications Surveys and Tutorials*, 2023 [153]. **In submission**.

– Yebo Feng. "Toward finer granularity analysis of network traffic." *Computer and Information Science, University of Oregon, Technical Report, AREA-202203-Feng*, 2022 [148].

I am the leading author of the above articles. The content of this chapter was written entirely by me, and I was responsible for conducting all the presented study and analysis.

2.1 Introduction

In the context of Internet, protocols and applications are usually built upon hierarchical models [360] (e.g.TCP/IP and OSI), where the communication functions of a telecommunication or computing system are categorized into several abstraction layers. Higher layers only encapsulate high-level methods, protocols, and specifications, operating with the support of lower layers [414]. With such design, programmers can easily develop interoperable Internet applications regardless of diverse underlying protocols and technologies. However, this convention also makes cross-layered network analysis feasible. As developers of higher layer applications usually only take higher-layer measures (e.g.encryption, anonymization, etc.) to preserve the user privacy regardless of leaving traceable patterns on lower layers, analyzers can capture network features from the lower layers to infer higher-layer knowledge in communication [317], even in the presence of message encryption. Such a process is called TA, a technique widely used in today's Internet.

TA has been studied for decades, with myriad systems, tools, and algorithms [353, 137, 299, 294, 303, 229] developed to serve different types of purposes, such as traffic measurement, traffic engineering, anomaly detection, and network surveillance. In early development of TA, traditional TA approaches were mainly designed for network traffic measurement/forecast [335, 236, 272], anomaly detection [56], and basic traffic classification [140]. These approaches are usually rule-based, statistics-based, or clustering-based, can separate traffic of different network protocols or conduct basic modeling of traffic flow changes. Later, with the adoption of cutting-edge data processing techniques and algorithms, such as harnessing the power of machine learning on big data, TA is able to deduce more information from network traffic data regarding application-layer activities, fine-grained user behaviors, and message content. For instance, researchers have developed advanced TA techniques to detect application-layer threats, infer the specific websites that people are visiting over HTTPS, or even dig users' private data from network-layer knowledge. We define such advanced TA techniques as FGTA, the process of application-layer behavior modeling, fine-grained user activities inferring, or traffic content decoding, only through link-layer or network-layer traffic data, with or without encryptions.

As a subset of TA, FGTA is mainly different from traditional TA in the following ways:

– The most notable difference is the goals of analysis. Traditional TA can coarsely distinguish or model traffic from different types of network device, protocols, or applications. However, FGTA aims to analyze traffic at a finer granularity, such as traffic from different application-layer activities (e.g.Twitter post vs. Tweeter read), different groups of application users (e.g.OSN bots vs. normal users), or different user content (e.g.the visiting website).

– The analysis pipelines of traditional TA and FGTA are usually different. FGTA, aiming at more granular information, usually takes the traditional

TA as a prerequisite step to "preprocess" the traffic before the final inference. For example, a FGTA approach that tries to identify the web page the user is visiting needs to first leverage traditional TA to extract all the web browsing traffic.

- As for analysis algorithms, most FGTA approaches depend on sophisticated modeling or classification methods, such as deep machine learning or high-dimensional clustering, to tackle the challenging fine-grained object identification tasks. While, traditional TA, dealing with easier tasks, can utilize a number of different analytical methods, such as rule-based, statistics-based, or soft-computing-based approaches.

With the increasingly complex Internet architecture, increasingly frequent transmission of user data, and the widespread use of traffic encryption [299], FGTA is becoming a more and more important research topic. Compared with traditional TA, FGTA can reveal more information from network traffic and can achieve high efficacy even in various complicated network environments [342]. Besides, as network traffic data become more easily accessible than before, the applicable scenarios of FGTA are more extensive compared with directly analyzing traffic content. Furthermore, FGTA is efficient and portable in discovering application-layer knowledge [297, 380, 151]. By analyzing a small amount of metadata or statistical information of traffic, FGTA can obtain almost the same level of visibility as decoding large amount of message content. Therefore, FGTA has a wide range of usage scenarios. As for network managements, FGTA can help measure application usage [337], detect complicated network intrusions or anomalies [49], investigate edge user experience [426], etc. As for the attacker side, FGTA can help eavesdrop on private information of users [433], model user

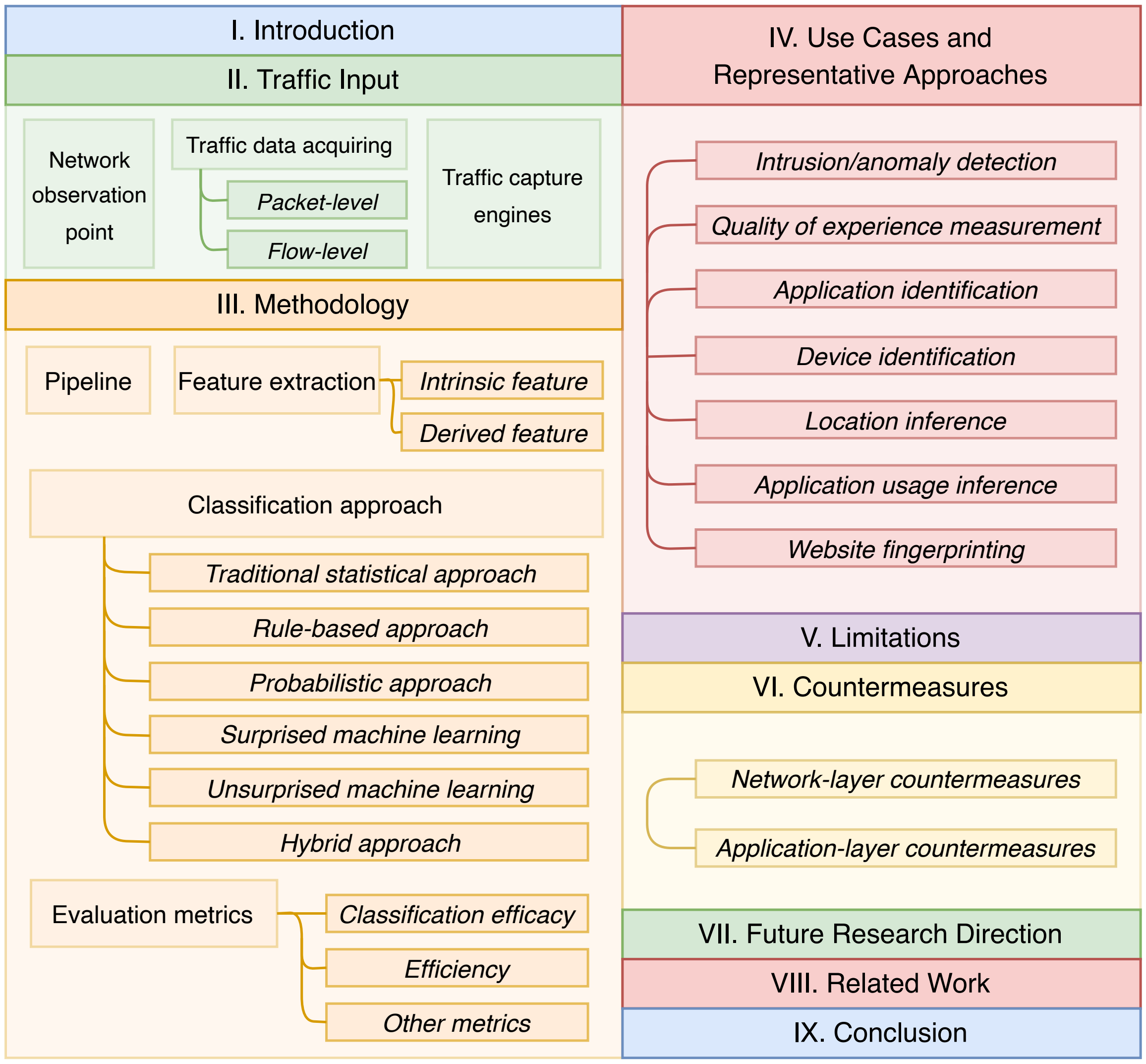


Figure 1. The organization of Chapter 2 and snapshots of proposed taxonomies.

behaviors [255], estimate user locations [52], etc. Studying FGTA is essential for comprehensive network inspection, safeguarding information transmission, and precise network configuration.

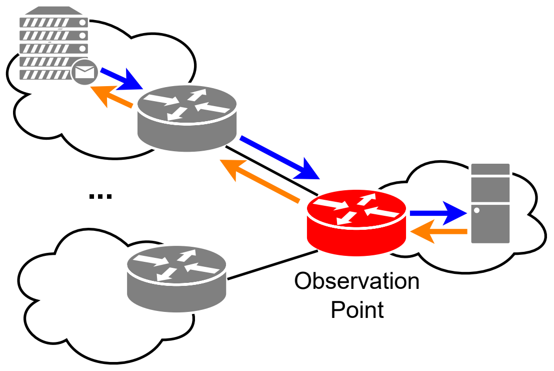
In this chapter, we examine the literature that deals FGTA. By including more than 230 citations, we first discuss the network traffic data and its collection in real-world environments. We then elaborate on frontier developments of FGTA, demonstrating and comparing different FGTA approaches' operating pipelines, classification approaches, and use cases. We further indicate the limitations and countermeasures to current FGTA approaches. In the end, based on our observations and reflections on this field, we propose several avenues for future research, thereby helping future academics and developers to advance FGTA. To our best knowledge, this paper is the first survey paper that focuses on FGTA and compares the state-of-the-art approaches in this field.

The rest of this chapter is organized as follows. After describing the input data of FGTA in Section 2.2, we discuss and summarize the methodologies of FGTA in Section 2.3. We elaborate on frontier developments of FGTA by their use cases in Section 2.4. We then point out the limitations of existing FGTA in Section 2.5 and introduce the countermeasures in Section 2.6 In the end, we propose some avenues for future research in Section 2.7 and conclude this report in Section 2.9. Figure 1 illustrates the organization of this survey paper and give snapshots of the proposed taxonomies.

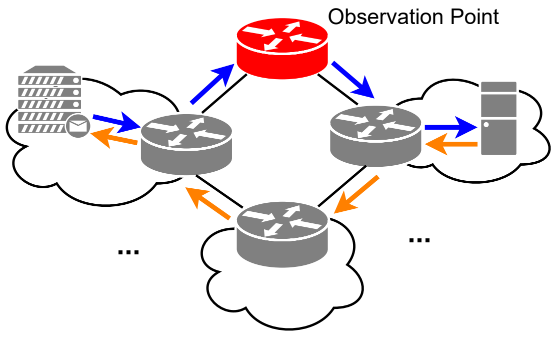
2.2 Traffic Input

Like traditional TA, FGTA approaches use network traffic data from some vantage points in the network as input to synthesize knowledge. Network traffic data refers to the information exchanged between devices on a computer network. Such data can be in diverse formats and include a wide range of information, such as communication logs, packet headers, and payload. The network traffic data is the inference object for all TA approaches. In this section, we compare different types of network traffic data and survey their capture engines. We also introduce these capture engines' deployments and application scenarios in FGTA.

2.2.1 Network Observation Point. The observation point of the traffic capture engine will significantly impact the integrity of the captured data



(a) The observation point is the gateway of the network. The traffic capture engine can collect bidirectional traffic data.



(b) The observation point is in the network. The captured traffic can be asymmetric.

Figure 2. Network visibility with different observation locations.

and the network visibility. Different observation points are suitable for different types of TA tasks.

The ideal observation point for most FGTA tasks is located at the gateway of a network (illustrated in Figure 2a), which enables them to capture both inbound and outbound traffic of the network. Such a bidirectional traffic dataset is suitable to infer the interactions between the observed network and rest of the Internet. However, analyzers cannot learn about the traffic in the rest of the Internet according to this dataset.

Sometimes, the observation point can be in the middle of the network (illustrated in Figure 2b), especially when the traffic capture engine is deployed by an ISP or IXP. In this case, the capture engine is able to collect a large amount of traffic that pass by it. However, it also raises the following concerns:

– Due to asymmetric packet routing [194], in-network observation point sometime may only capture traffic in one direction (illustrated in Figure 2b).

– It cannot guarantee the robustness of captured traffic because of the deployment of various traffic engineering techniques [398, 42]. The routing path for any packet can be dynamic in today's networks.

Therefore, in-network-based observation points may be more suitable for traditional TA tasks such as Internet measurement and network-layer anomaly detection. As for FGTA, many approaches (e.g., user behavior inference, website fingerprinting) prefer to use the gateway-based observation point to capture more complete traffic data from endpoints. However, wherever the observation point is located, it is difficult to capture all the relevant traffic in the network.

To capture comprehensive traffic data from the network with complex topology, we can deploy multiple observation points at different vantage points if conditions permit. By using a pool of metering processes to collect network packets at multiple observation points, optionally filter them and aggregate information about these packets, a traffic exporter can gather each of the observation points together into an observation domain and sends this information to a traffic capture engine [375]. Then we can obtain relatively comprehensive network traffic data without redundancy. However, this approach is expensive to deploy and not always feasible due to real-world constraints.

2.2.2 Traffic Data Acquiring. Since the birth of the Internet, various traffic capture engines have been developed to log traffic information. TA approaches can further leverage these "log information" to measure network events, detect anomalies, and analyze network behaviors. Based on different information captured, these traffic capture engines can be classified into either packet-level or flow-level [353].

2.2.2.1 Packet-level capture.

Packet-level capture is widely used in local networks and endpoint devices. As its name states, it copies or makes a snapshot of all the network packets that pass by the network interface and forwards the collected data to a collector. The agent that takes charge of the capture is called a packet-level traffic capture engine or a “sniffer”, which can be either software-based (e.g.Snoop [10], Wireshark [291], etc.) or hardware-based (e.g.Sniffer InfiniStream [4]). It can be as simple as an IP table rule on a route that copies all the traffic to a cloud disk besides normal forwarding.

Packet-level capture can collect raw network traffic, containing both packet headers and packet payloads. Theoretically, it can support all types of FGTA tasks because it basically logs all the information flowing on networks. However, in most cases, packet-level traffic capture might not be the right solution to deploy for the following reasons:

- Packet-level traffic capture is expensive, not only because the interface needs to copy all the packets that pass by it, but also because the interface needs to forward all the captured traffic to an analysis node through a link. All these operations will double the workload of the network interface and occupy a considerable amount of link bandwidth. Packet-level traffic capture is therefore not scalable.

- The information contained in packet-level traffic data is sometimes an “overkill” for TA, as many TA approaches only require statistical information from the packet headers to complete the analysis. Moreover, user messages, website content, and video streaming are usually contained in packet payloads in encrypted forms, making most information captured in packet-level traffic meaningless for all TA approaches.

– Packet-level traffic may contain sensitive information (i.e.payload) of users. Thus, network service providers are cautious about capturing and analyzing such data.

2.2.2.2 Flow-level capture. To address the aforementioned issues of packet-level traffic captures and make traffic capturing affordable, scalable, and practical for network service providers, researchers and developers have proposed myriad flow-level traffic capture engines.

In flow-level traffic capture systems, the capture engines no longer copy or make snapshots of each packet, instead, they first aggregate relevant packets into a flow and then capture metadata or statistical information to represent that flow. Here, the concept of flow has been around for a long time. Typically, a flow can be identified by either a 5-tuple (i.e., source IP address, source TCP/UDP port, destination IP address, destination TCP/UDP port, and IP protocol) or a 3-tuple (i.e., source IP address, destination IP address, and IP protocol) [54]. However, with the development of flow capture engines, researchers have proposed many other formal and informal definitions of network traffic flows (e.g.RFC 2722 [78], RFC 3697 [313], RFC 3917 [311], etc.). In this paper, we define a network traffic flow as a sequence of relevant network packets from a source to a destination for the same application. In most instances, the network system will process packets within a flow in the same manner. Besides, each application-layer behavior will generate one or multiple flows in both directions.

By capturing traffic at flow-level, traffic capture engines no longer suffer from high system overhead and high bandwidth usage. Figure 3 illustrates the workflows of a network interface with and without Cflowd as the flow-level traffic captured engine [268]. Unlike packet-level traffic capture that will copy and forward

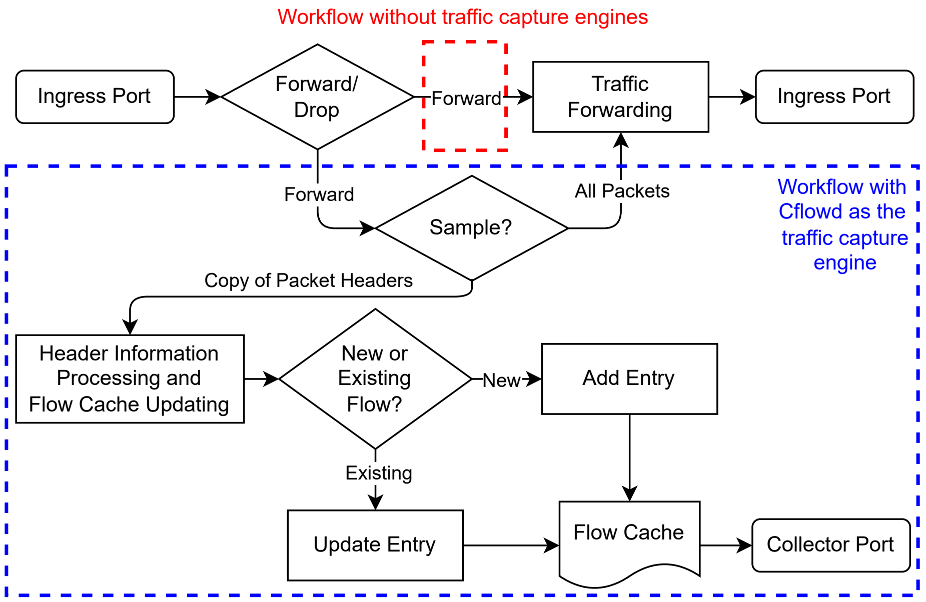


Figure 3. Workflow of a network interface when Cflowd serves as the traffic captured engine.

any packet entirely to the collector port, flow-level traffic capture only copies information from headers to assemble traffic flows. The volume of data to process is then largely reduced in such a procedure. According to existing literature [201], NetFlow, the most frequently used flow-level traffic capture engine, only creates 1-1.5% of throughput (without sampling) on the interface it is exported on [242]. With a great deal of data reduction, network administrators can store, process, inspect and analyze large amounts of network data efficiently. Furthermore, when combining this procedure with packet sampling, it becomes feasible to capture and store traffic flows at an ISP or IXP scale, thereby extending the usage scenarios of TA. As we can see from a study, NetFlow only occupies around 15% of the router/switch's CPU load when capturing sampled network traffic [96]. Compared with packet-level traffic capture that sometimes may double the system overhead

Table 1. Comparisons of selected widely-used traffic capture engines (●: fully support; ◐: partially support; ○: not support.).

Traffic Capture Engine	Data Captured	Granularity	Open or Proprietary	Layer (OSI)	Hardware Acceleration	Sampling
SNMP [189]	High-level statistical information about the interface.	Flow-level (aggregated)	Open	2, 3	○	○
IPFIX [103]	Metadata and statistical information about the flow.	Flow-level	Open	3, 4	●	●
NetFlow v9 [102]	Metadata and statistical information about the flow.	Flow-level	Proprietary	3, 4	●	●
NetFlow v5 [8]	Metadata and statistical information about the flow.	Flow-level	Proprietary	3, 4	●	●
Argus [2]	Metadata and statistical information about the flow.	Flow-level	Open	2, 3, 4	●	○
sFlow [307]	Complete packet headers and partial packet payloads.	Packet-level	Partially Open	2 - 7	●	●
Tcpdump [11]	Network information pass through the observation point.	Packet-level	Open	2 - 7	○	○
Wireshark [291]	Network information pass through the observation point.	Packet-level	Open	2 - 7	○	○
PF_RING [33]	Network information pass through the observation point.	Packet-level	Open	2 - 7	●	◐
Netmap [325]	Network information in the memory of the observation point.	Packet-level	Open	2 - 7	●	◐

and link usage, flow-level traffic capture is a huge improvement regarding efficiency and deployability.

However, the shortcoming of flow-level traffic capture is also obvious—it will decrease the visibility of the network traffic because people only see metadata and aggregated statistical information about the traffic rather than each packet. This is especially troublesome for FGTA as many approaches require at least inter-packet information (e.g., packet interval time). To make up for this, we can shorten the lifecycle for each flow in traffic capture engines to let them generate flows more frequently, thereby increasing the network visibility.

2.2.3 Widely used traffic capture engines.

Here, we introduce widely-used traffic capture engines in academia and industry (Table 1 shows comparisons of them).

2.2.3.1 Packet-level traffic capture engines.

Back in the early days of Internet, developers had realized the importance of capturing network packets for troubleshooting. Thus, Tcpdump [11], a software-based packet-level traffic capture engine (sniffer), was proposed in 1988. It allows users to store and display TCP/IP and other packets being transmitted or received over a network. Nowadays, Tcpdump has been ported to several operating systems (e.g.Unix with libpcap library, Windows with WinPcap) and is still frequently used in network studies. Similar software-based sniffers were also proposed to meet different needs. For example, Snoop [10], a simple packet capture tool that is bundled on Solaris operating system; Wireshark [291], a free packet capture and analysis software that not only supports multiple operating systems (e.g.Linux, Solaris, Windows, FreeBSD, Mac OS, etc.), but also comes with a user-friendly interface; PF_RING [33], a high speed packet capture library that can turn a commodity PC into an efficient and cheap network measurement box suitable for both packet capture and TA. As for routers and switches, traffic mirroring [315, 362, 397] is also well-studied, with many software or hardware-based approaches [4, 27] proposed to support real-time packet capture for enterprise-level networks.

However, as capturing the entire packet is expensive and sometimes impractical, people began to make a snapshot of each packet rather than storing it entirely. The most frequently-used approach is sFlow [307], an industrial method (defined in RFC 3176 [307]) originally developed by InMon Inc., to capture packet-level snapshot from switches and routers. Compared with previous packet-level traffic capture engines, sFlow has the following features, making it the ideal input for most FGTA approaches:

- Without capturing the entire packet, sFlow can just copy the first N bytes of a packet to save computing and transmission resource. This is especially useful for TA tasks as packet payloads are useless in such scenarios but the entire packet headers are still preserved for fine-grained analysis.
- As an industrial standard, sFlow is compatible on many different platforms of network switches and routers and utilizes a dedicated chip built into the devices to operate, which removes the burden of the CPU and memory of the router or switch when capturing the traffic.
- By introducing time-based or packet-based sampling techniques, sFlow can capture traffic on all interfaces simultaneously at wire speed.

Therefore, sFlow can reach a good balance between data integrity and velocity—being able to capture all the packet headers and simultaneously create less burden on the router or switch.

2.2.3.2 Flow-level traffic capture engines. Flow-level traffic capture engines also have a long history. Back in 1984, the Audit Record Generation and Utilization System (Argus flow [2]) was proposed as the first implementation of network flow monitoring, and is still an ongoing open source network flow monitor project now. Argus can monitor all network traffic, including Internet Protocol (IP) traffic, data plane, control plane and management plane. It captures much of the packet dynamics and semantics in each flow, providing reachability, availability, connectivity, duration, rate, load, delay metrics for all network flows. It also captures most attributes that are available from the packet headers [413]. Later, in 1988, Simple Network Management Protocol (SNMP) [189] was proposed as a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). Unlike Argus flow that provides rich information about ongoing traffic, SNMP only provides statistical information per interface, such as link utilization, interface bandwidth, and some other information if the device provides. SNMP is thus less applicable in TA compared with Argus, especially in the domain of FGTA.

With rapid development and popularization of the Internet, the industry had realized the importance of flow-level traffic capture engine and many solutions were proposed. The most typical example is NetFlow [102], so far the most widely-used flow-level capture engine with many TA approaches built upon. Just like Argus, NetFlow uses a flow record to represent a set of packets. However, unlike Argus, which is a bidirectional monitoring approach, NetFlow is a unidirectional flow monitor, reporting flow information of each direction of conversations independently. This feature allows NetFlow to have a finer granularity than Argus. Since NetFlow was developed by Cisco, it is bundled with most Cisco routers and switches, making it the object of imitation of the entire industry. Following NetFlow, many similar systems were proposed by both research institutions and commercial companies, such as Cflowd [268], J-Flow [114], NetStream [209], Remote Network Monitoring (RMON) [385], etc. NetFlow itself also has evolved into different variations. The most famous one is Internet Protocol Flow Information Export (IPFIX) [103], an IETF protocol built upon NetFlow v9.

The most recent development of traffic capture and traffic handling have been mainly focusing on the velocity issue. Researchers have proposed multiple approaches to capture large volume of network traffic at line speed without having

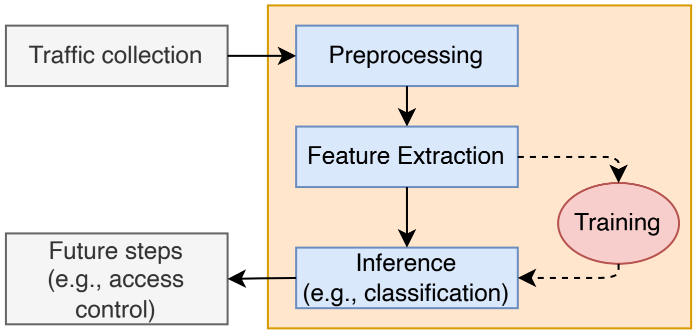

Traffic collection
Preprocessing
Feature Extraction
Training
Inference (e.g., classification)
Future steps (e.g., access control)

Figure 4. A general data processing pipeline for FGTA.

any effect on data plane. For example, Netmap [325] a memory-based framework that enables commodity operating systems to handle millions of packets per seconds without the support of custom hardware; eXpress Data Path (XDP) [203], a fast programmable packet processing approach based on the operating system kernel, supports high speed packet logging and processing; hXDP [79], an efficient software network packet processing approach written in extended Berkeley Packet Filter (eBPF) on Field Programmable Gate Arrays (FPGA) network interface controllers (NICs); NetSeer [445], a flow event telemetry (FET) monitor, which aims to discover and record all performance-critical events on the programmable data plane. However, those approaches do not change the pipeline of TA or FGTA, as they only make it faster to capture and handle network traffic.

2.3 Methodology

In this section, we delve into the methodology of FGTA and explore this field from the perspectives of data processing pipelines, feature extraction approaches, classification approaches, and evaluation metrics. These components are integral to the success of FGTA and play a crucial role in achieving accurate results.

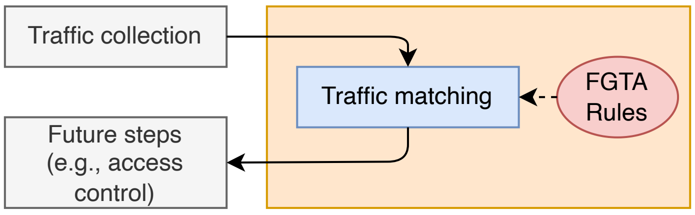


Figure 5. Example of a simplified FGTA data processing pipeline.

2.3.1 Pipeline. The process of generating fine-grained analysis results from raw network traffic collected from network infrastructures typically involves several necessary steps. These data processing procedures are known as the FGTA pipeline. Different FGTA approaches may have different pipelines, with different steps and different orders. In this subsection, we discuss three types of FGTA pipelines (illustrated in Figure 5, 4, and 6).

Figure 4 illustrates the most commonly-used pipeline for FGTA. Regardless of whether the input traffic is in flow-level or packet-level format, it usually cannot be directly processed by analysis algorithms. Therefore, the first step of the FGTA pipeline is usually to preprocess the raw traffic data. The preprocessing step typically involves the following tasks:

– **Data decoding:** the raw network traffic data is usually encoded in a format that is not easily processable (e.g., binary format or encrypted form). This task converts the raw traffic data into a readable and processable form.

– **Data cleaning:** the raw traffic data may contain some noise, invalid data, or control messages. This task extracts only the valuable data for further analysis.

– **Data refactoring:** this task refactors the raw network traffic data and make it suitable for the subsequent analysis or maintenance. For example, indexing the raw traffic data by socket pairs, or converting the flow records to a B tree structure [105].

- **Other tasks necessary for subsequent steps:** depending on different FGTA pipelines, there may be other tasks necessary for subsequent steps. For example, extracting marked packets from the raw traffic data, anonymizing the raw traffic data for General Data Protection Regulation (GDPR) compliance [383], or compressing the data for efficient storage.

After the preprocessing step, FGTA approaches usually move to feature extraction, which refers to the process of selecting and transforming raw network traffic data into a set of relevant features that are suitable for machine learning, inference, or other analysis steps. For both traditional TA and FGTA approaches, the feature extraction is a particularly important step for representing the ongoing network events and achieving accurate results. We further discuss more details about feature extraction in Section 2.3.2.

After relevant features are extracted, FGTA approaches are typically ready for inference. The inference goals of these approaches can vary, including identifying specific network events, classifying traffic flows based on different application behaviors, or detecting network anomalies. We further discuss the use cases of FGTA approaches in Section 2.4. The inference results of FGTA approaches can be used for a variety of purposes, including network monitoring, access control, device management, data center protection, etc. However, regardless of the inference goal, the inference step of FGTA always operates in the form of fine-grained classification. For example, classifying outlier traffic flows from normal traffic flows (i.e., anomaly detection), or classifying traffic flows according to different applications (i.e., application identification). Therefore, we use the

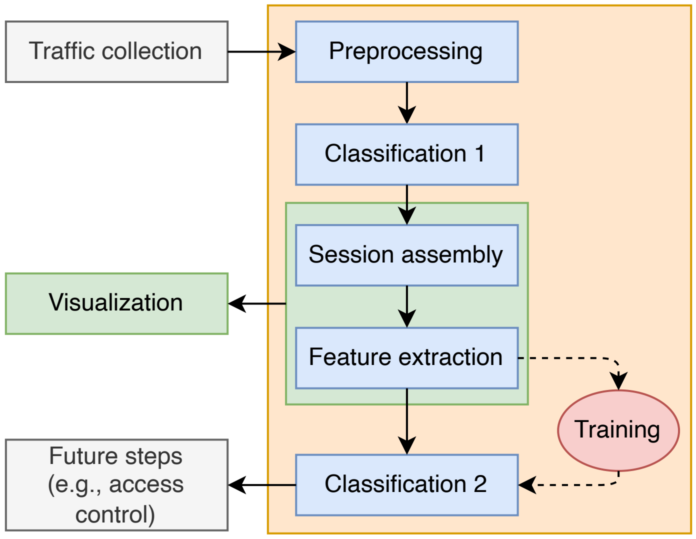


Figure 6. Example of a more complicated FGTA data processing pipeline.

term *classification* to refer to the inference step of FGTA approaches. Section 2.3.3 discusses the classification approaches used in FGTA.

The previously mentioned pipeline outlines the general steps for FGTA approaches. However, depending on the specific goals, system design, and operational environments, the FGTA pipeline can be simplified or extended, with specific steps omitted or added.

Figure 5 illustrates a simplified FGTA pipeline, where the raw network traffic data is directly used for rule-based traffic pattern matching. A short data processing pipeline is very efficient to operate and can still generate accurate results if the pre-defined matching rules are effective. It is useful when target traffic pattern is distinct or well-defined (i.e., location inference [52]). However, this short pipeline is not suitable for most of the FGTA approaches, as they often require additional steps to more thoroughly analyze the traffic data for generating fine-grained, application-layer classification results.

To infer high-level, fine-grained information from content-agnostic network traffic data, most FGTA approaches employ more complicated pipelines to mine hidden knowledge. Figure 6 illustrates such an example. Many application usage inference approaches apply similar pipelines [156, 255, 152] because they need to extract features and classify traffic for multiple times at different phases to derive detailed user behavior information of specific applications. The sample pipeline include two different classification steps, with one for narrowing down the analysis scope and the other for generating fine-grained classification. More importantly, this pipeline re-assemble traffic flows into sessions (some papers may call them transactions [151] or bursts [343, 337]) before extracting features for the final classification. This step is very helpful for digging fine-grained behavior information from the traffic data because the target network behavior or event usually consist of multiple packets or traffic flows. Simply analyzing the network traffic flow by flow or packet by packet may not be able to capture the whole picture of the ongoing network events. Therefore, session assembly is used to aggregate adjacent, relevant, or similar traffic data into an analysis unit, which is a more representative data structure to present the ongoing network events and makes it possible to infer fine-grained application-layer information. Figure 7 illustrates an example of session assembly [152], where flow records are divided into flow points and then aggregated into sessions according to the traffic density. Based on current literature, the following approaches are commonly used for session assembly:

- **Time-based session assembly:** this approach aggregates traffic flows into sessions based on the timing or the intervals of ongoing network traffic.

- **Clustering-based session assembly:** this approach utilize clustering algorithms to group traffic flows into sessions.

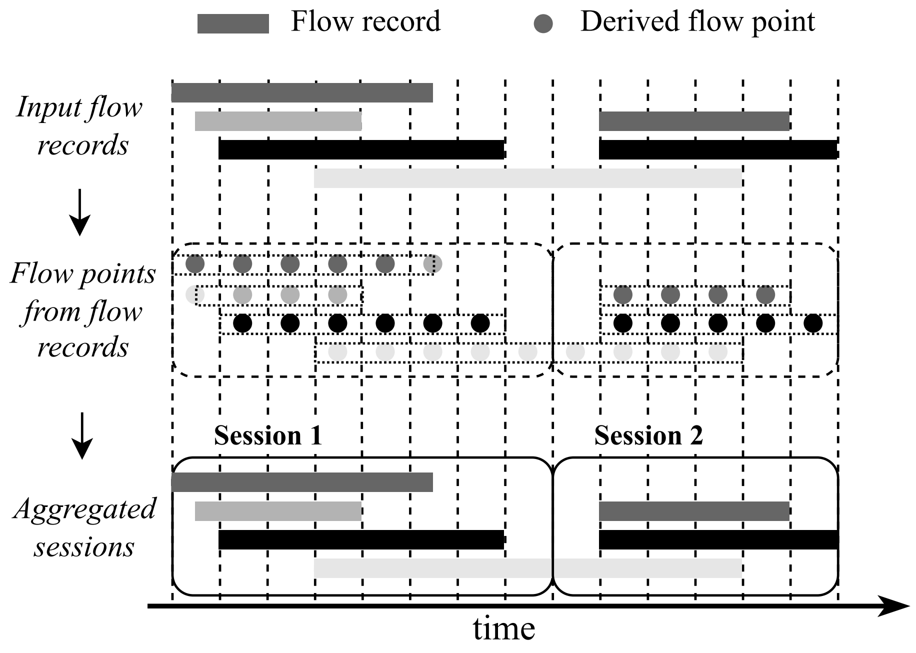


Figure 7. Example of a session assembly procedure, where relevant flow records are aggregated into a traffic session to represent a network event.

– **Index-based session assembly:** this approach aggregates traffic flows into sessions by specific indexes (e.g., socket pair, packet ID ranges, time to live (TTL), etc).

– **Rule-based session assembly:** this approach aggregates traffic flows based on pre-defined rules (e.g., rules on the hash value of packet payload, rules on TCP flags, etc.).

After sessions are assembled, representative features can be properly extracted for fine-grained classifications.

2.3.2 Feature extraction.

Feature extraction is a term refers to the process of selecting and generating relevant features from the raw data in order to create a representation that can be used for machine learning, statistical modeling, or other analysis procedures [185]. In the context of FGTA approaches, feature

Table 2. Examples for intrinsic and derived features.

Category	**Example**	
Intrinsic feature	Flow-level	Packet-level
	Flow size, number of packets, AS number, protocol type, flow duration, etc.	TCP flag, ToS, packet size, packet interval, first n bytes of the payload, etc.
Derived feature	Flow/packet-based	Session-based
	Interval deviation, size deviation, interval distribution, inbound/outbound packet ratio, packet similarity, etc.	Session duration, density distribution, session image, normalized session vector, round-way communication number, etc.

extraction involves inspecting network traffic data to identify relevant features that can be used for the corresponding classification tasks. This process typically involves techniques such as packet inspection, data fusion, and statistical modeling to identify and derive patterns or characteristics in the data that are relevant to the specific inference goal. The resulting set of features is then used as input to the classification model.

Due to the nature of network traffic collection, all the extracted features can be categorized into two types: intrinsic features and derived features. Intrinsic features are directly contained in the raw network traffic data, such as packet length, packet header fields, etc. The process of fetching intrinsic features is simple and straightforward. The analysis system can directly select, slice, or generate intrinsic features from the raw data, requiring little to no additional processing. On the other hand, derived features are not directly contained in the raw network traffic data. They are generated by applying some data processing techniques to the raw data, such as statistical modeling, feature transformation, information assembly, data fusion, etc.

Table 2 lists some typical examples of intrinsic and derived features. Different features are suitable for different FGTA tasks. Typically, some relatively easy FGTA tasks may only require intrinsic features to operate. For example, some application identification or anomaly detection approaches can generate accurate results by directly inputting intrinsic features. Because the network traffic of such applications or anomalies can already be distinguishable by intrinsic features [44, 366, 380]. However, some more complex FGTA tasks may require derived features for finer granularity analysis, especially for tasks that infer detailed, application-layer user behaviors [337, 321, 152]. The target network traffic of these FGTA tasks is less distinguishable and may only show obvious patterns with sophisticated feature engineering techniques. We discuss more details about suitable features for different FGTA tasks in Section 2.4.

Although derived features are more powerful than intrinsic features in mining fine-grained information from network traffic data, they are also more complex and expensive to generate. One may need to apply sophisticated data processing techniques, such as traffic buffering, data fusion, session assembly, statistical modeling, etc., to fetch these derived features. Such processes are time-consuming and may require significant computational resources. As time-sensitive tasks, it is vital for FGTA procedures to be efficient and scalable, thereby outputting analysis results in a timely manner. Thus, carefully selecting and generating necessary features is a critical step for all FGTA approaches.

2.3.3 Classification approach. The key step of FGTA is to classify the target network traffic from others. Designing a proper classification approach determines the efficacy and performance of FGTA approaches. In many cases, constructing a classification model requires labeled data. In the context of FGTA, labeled data refers to network traffic data that has been manually labeled or annotated with ground truth information. This ground truth information typically includes information such as the application type, user behavior type, or whether the traffic is generated by malicious behavior or not. Obtaining labeled data can be a challenging and resource-intensive process. It typically requires a significant amount of manual effort and expertise to accurately label network traffic data. Researchers may be able to automate the labeling process with the help of other state-of-the-art classification approaches, but the accuracy of labels may not be ideal [183]. On the other hand, some classification approaches can be trained without labeled data or with other forms of prior knowledge. In the remaining of this subsection, we discuss the classification approaches that are commonly used in FGTA approaches (summarized in Table 3).

2.3.3.1 Traditional statistical approach. Traditional statistical approaches leverage statistical properties, statistical models or some other mathematical methods to identify subtle differences or patterns in different groups of network traffic [300]. Typical examples of statistical approaches include distribution fitting [155], logistic regression [218], linear regression [49], etc. Traditional statistical approaches are widely used in traditional TA tasks because they are explainable, easy to implement, usually efficient to operate, and good at tackling relatively easy tasks. However, as FGTA tasks becoming more and more challenging, traditional statistical approaches are not sufficient to identify subtle differences in network traffic. Thus, traditional statistical approaches are gradually replaced by more sophisticated classification approaches, such as machine learning approaches. Still, traditional statistical approaches are commonly used in feature extraction, pre-analysis, and pre-classification. For example, many FGTA

Table 3. Summary of widely-used classification approaches in FGTA.

Category	Description	Representative algorithms/ approaches	Pros	Cons	Use in FGTA	Reference
Traditional statistical approach	Leverage statistical properties or statistical models for FGTA.	Distribution fitting, regression, variance matching, etc.	Explainable, easy to implement, and efficient.	Poor efficacy.	Limited	[155, 218, 49]
Rule-based approach	Utilize a set of pre-defined rules to locate the target network traffic group.	Session signatures, traffic thresholds, predefined packet header fields, etc.	Explainable, easy to implement, and efficient.	Hard to define rules, poor efficacy, and poor flexibility.	Limited	[426, 74, 297]
Probabilistic approach	Approaches based on probability theories to analyze network traffic.	Bayesian classifier, Markov model, HMM, etc.	Flexibility, adaptability, and ease of use.	Complexity, sensitivity to assumptions, limited accuracy, and relatively poor explainability.	Popular	[82, 166, 44]
Supervised machine learning	ML methods that rely on learning knowledge from labeled data.	KNN, SVM, LSTM, etc.	Great efficacy, ease of use, and good flexibility.	Limited explainability, overfitting, dependency on labeled data, and limited scalability.	Most popular	[408, 37, 258]
Unsupervised machine learning	ML methods that do not require labeled training data and can discover patterns and relationships in the data on its own.	K-means, PCA, DBSCAN, etc.	Discovering unknown patterns, flexibility, no labels needed, and no training time.	Limited result interpretability, limited efficacy, poor scalability in inference, and overfitting.	Popular	[48, 152, 345]
Hybrid approach	Combine multiple classification approaches for better performance.	Ensemble model, semi-supervised machine learning, combining statistical approaches with rule-based approaches, etc.	Inherits advantages of multiple classification approaches.	Complicated to design, and computationally expensive.	Popular	[380, 155, 345]

approaches use traditional statistical approaches to narrow down the analysis scope before fine-grained analysis, thereby reducing the computational complexity of the subsequent procedures [155].

2.3.3.2 Rule-based approach. Rule-based approaches are based on a set of pre-defined rules that are manually designed by experts to locate the target network traffic group [134]. Before defining the classification rules, the experts usually need a thorough understanding of the target network traffic and the network environment. Typical examples of classification rules include session signatures [426], traffic thresholds [74], pre-defined packet header fields [297], etc. Similar to traditional statistical approaches, rule-based approaches are explainable, easy to implement, and efficient to operate, thereby being widely used in traditional TA tasks. However, in the era of FGTA, the analysis tasks are in finer granularity and becoming more and more challenging. Thus, the pre-defined rule sets are becoming larger, more complex, making them more difficult to define, verify, and maintain. Moreover, the rule-based approaches are not able to adapt to the dynamic network environment, which is a common feature of modern networks. Therefore, in recent trends, rule-based approaches are less used in FGTA tasks. But they are still powerful tools in some specific FGTA tasks, pre-classification, and accelerating the analysis process.

2.3.3.3 Probabilistic approach. Probabilistic approaches are based on probability theory and statistical inference to identify the target network traffic group [47]. They model the traffic data probabilistically for classification tasks. For instance, typical probabilistic approaches like Bayesian classifier [324], Markov model [121], or hidden Markov model (HMM) [312] are widely used to model network traffic first. These models can then be utilized to identify traffic patterns of specific applications, protocols, anomaly, or behaviors. Benefitting from the following advantages, a variety of FGTA approaches have been proposed based on probabilistic approaches to tackle different FGTA tasks [82, 166, 44]:

– Flexibility: probabilistic approaches can be used to model a wide range of traffic patterns and behaviors. Besides, they can tolerate noise and uncertainty in the data, making them powerful tools for analyzing complex and heterogeneous traffic data.

– Adaptability: probabilistic approaches can be easily adapted to changes in traffic patterns over time, allowing them to detect new or previously unseen threats.

– Ease of use: with supports of various libraries, probabilistic approaches are relatively easy to implement and does not require extensive domain knowledge or expertise.

However, probabilistic approaches feature the following disadvantages, resulting limited performance and application especially in complicated FGTA tasks (e.g., user behavior inference):

– Complexity: probabilistic approaches are usually computationally expensive, especially when the network traffic data is large and complex.

– Sensitivity to assumptions: probabilistic approaches are sensitive to the assumptions (labels) made during model training or development, and incorrect assumptions can lead to inaccurate results.

– Limited accuracy: probabilistic approaches may not achieve the highest accuracy compared to other more advanced methods, such as deep learning, in some scenarios. Also, they are relatively weak in handling nondiscrete data.

– Explainability: probabilistic approaches may not provide as much interpretability as other methods, making it difficult to understand how the models arrived at their conclusions.

2.3.3.4 Supervised machine learning. Supervised machine learning is a widely used machine learning method that can be applied to almost any FGTA tasks with reliable prior knowledge [342]. In supervised machine learning, a classifier is trained using a labeled training dataset that includes known classification labels. The trained classifier is then used to classify or detect anomalies in new traffic data. Supervised machine learning approaches typically involve two main phases: training and inference. During the training phase, the classifier is trained on the labeled dataset (i.e., labeled network traffic) to learn the relationship between the input features and the classification labels. The inference phase involves using the trained classifier to infer the classification labels of ongoing network traffic.

With decades of development, researchers have proposed a variety of supervised machine learning approaches [220], from traditional machine learning methods, such as k-nearest neighbor (KNN), decision tree, Support Vector Machine (SVM), to advanced deep learning methods [241, 180], such as multi-layer perceptron (MLP), recurrent neural network (RNN), long short-term memory (LSTM). Each of the proposed supervised machine learning approaches has its own advantages and disadvantages, making them suitable for different FGTA tasks. Selecting the most suitable supervised machine learning approach is the key to designing an effective ML-based FGTA approach. We discuss more details about the ML algorithm select by use case in Section 2.4.

Overall, due to the following advantages, supervised machine learning approaches are the most widely used approaches in FGTA [49, 408, 37, 258]:

- High accuracy: supervised machine learning can achieve high accuracy in FGTA, especially when compared to other methods.

- Ease of use: on the one hand, supervised machine learning approaches are relatively easy to implement, with supports of various libraries and tools [34, 302, 216]. On the other hand, they do not require extensive domain knowledge or expertise to manually identify distinguishable rules or patterns.

- Flexibility: supervised machine learning approaches can be used to model a wide range of traffic patterns and behaviors.

However, supervised machine learning approaches also have many shortcomings that limit their performance and use cases:

- Limited explainability: many supervised machine learning algorithms, such as deep neural network (DNN), can be difficult to interpret, which can limit their usefulness in some applications, especially in anomaly or attack detection.

- Overfitting: supervised machine learning models can overfit to the training data, which can result in poor performance on new data.

- Dependency on labeled data: supervised machine learning requires high-quality labeled training data, which can be time-consuming and expensive to collect, making them less effective than unsupervised or semi-supervised methods in some cases.

– Limited scalability: many supervised machine learning approaches may not scale well to extremely large or complex datasets. Both the training and inference phases may be computationally expensive.

2.3.3.5 Unsupervised machine learning. Unlike supervised machine learning, unsupervised machine learning is a machine learning method that does not require labeled training data and can discover patterns and relationships in the data on its own [46]. Unsupervised machine learning algorithms typically involve clustering [378] or dimensionality reduction [379] techniques that can help identify similarities and differences between traffic flows. These algorithms do not directly output labeled classification results, but can be used to group similar traffic flows together or identify anomalous traffic flows that do not fit into any of the existing clusters. Widely used unsupervised machine learning algorithms include K-means [190], DBSCAN, principal component analysis (PCA) [135], hierarchical clustering [283], etc.

Unsupervised machine learning algorithms feature the following advantages in FGTA:

– Discovering unknown patterns: unsupervised machine learning approaches can identify previously unknown patterns and behaviors in the traffic data, which can be useful for detecting new or emerging threats.

– Flexibility: unsupervised machine learning approaches can be more flexible and adaptable than supervised machine learning as they do not require labeled data, making them easy to work with a wide variety of traffic datasets.

- No training time: unsupervised machine learning approaches usually takes zero training time, making them more efficient than supervised machine learning approaches regarding model development.

They also inevitably have the following disadvantages:

- Limited result interpretability: interpreting the results of the clustering or dimensionality reduction algorithms used in unsupervised machine learning can be difficult without prior domain knowledge.

- Limited accuracy: unsupervised machine learning may not achieve the same level of accuracy as supervised machine learning, especially when dealing with complex or noisy traffic datasets.

- Scalability in inference: although taking no time for training, some unsupervised machine learning algorithms are computationally expensive in the inference phase, which can limit their scalability.

- Overfitting: unsupervised machine learning models can also suffer from overfitting or underfitting, which can result in poor performance on certain datasets.

In conclusion, unsupervised machine learning is a powerful tool for FGTA, but it may have limitations in terms of result interpretability and accuracy. Due to such natures, unsupervised machine learning approaches are not widely used in tasks such as anomaly detection and attack detection [48, 152].

2.3.3.6 Hybrid approach. Hybrid approaches combine the advantages of multiple classification approaches to achieve better adaptability, explainability, or performance. People can use different approaches as different procedures in the FGTA pipeline, enhancing the feature extraction or pre-classification phase, or simply use an ensemble classification model to increase the robustness. For example, combining supervised and unsupervised machine learning approaches for semi-supervised traffic classification [380], combining statistical approaches with machine learning approaches better FGTA performance [155], or utilizing a variety of approaches in the FGTA pipeline for more comprehensive attack coverage [345].

Although hybrid approaches can usually achieve better performance, they are more complex to design and assemble. Besides, hybrid approaches are usually more computationally expensive than using single approaches.

2.3.4 Evaluation metrics. In FGTA, evaluation metrics are important measures of the performance, efficiency, and usability of proposed approaches. In this section, we discuss some commonly used evaluation metrics.

2.3.4.1 Classification efficacy. The most important evaluation metric for FGTA is the classification efficacy, which measures the accuracy of the proposed approach in classifying the target traffic flows. The classification efficacy is well-defined in the domain of data mining [245]. It can be measured by true positive rate (TPR), true negative rate (TNR), positive predictive value (PPV), negative predictive value (NPV), false negative rate (FNR), false positive rate (FPR), false discovery rate (FDR), false omission rate (FOR), F1 score (F1), accuracy (ACC), receiver operating characteristic (ROC), area under the curve (AUC), etc. Table 4 lists the calculations and descriptions of these

Table 4. Widely used classification efficacy metrics for FGTA, where TP denotes the number of true positives, TN denotes the number of true negatives, FP denotes the number of false positives, and FN denotes the number of false negatives.

Category	**Description**	**Calculation**
TPR	The probability that an actual positive will test positive.	$\frac{TP}{TP+FN}$
TNR	The probability that an actual negative will test negative.	$\frac{TN}{TN+FP}$
PPV	The probability that an item with a positive test result is truly positive.	$\frac{TP}{TP+FP}$
NPV	The probability that an item with a negative test result is truly negative.	$\frac{TN}{TN+FN}$
FNR	The probability of positives which yield negative outcomes.	$\frac{FN}{FN+TP}$
FPR	The probability of negatives which yield positive outcomes.	$\frac{FP}{FP+TN}$
FDR	The probability that an item with a positive test result is truly negative.	$\frac{FP}{FP+TP}$
FOR	The probability that an item with a negative test result is truly positive.	$\frac{FN}{FN+TN}$
F1	The harmonic mean of precision (PPV) and recall (TPR).	$\frac{2TP}{2TP+FP+FN}$
ACC	How close a given set of analysis results are to their true value.	$\frac{TP+TN}{TP+TN+FP+FN}$
ROC	A graph showing the performance of a classification model at all classification thresholds.	Through TPR and FPR.
AUC	The area under the entire ROC curve.	Through ROC.

metrics. For specific FGTA tasks, some metrics may be more important than others. For example, in anomaly detection, FPR and FNR are more important than PPV and NPV because FPR determines the false alarm rate, reflecting the usability of proposed approaches, and FNR determines the miss rate, reflecting the detection effectiveness of proposed approaches. While in tasks such as webpage fingerprinting, PPV and NPV are more important than FPR and FNR because PPV and NPV are more relevant to the classification efficacy.

2.3.4.2 Efficiency. Efficiency is another important evaluation metric for FGTA, which is usually measured by the time cost of finishing analyzing a certain amount of traffic flows by the proposed approach. In addition to accuracy and other performance metrics, the time cost can have a significant impact on the proposed approach's practicality and applicability. In real-world scenarios, the majority of FGTA tasks are performed on a large amount of traffic flows in real time. Therefore, the bottom line of these FGTA tasks is to reach the line speed in processing traffic flows. Here, the line speed refers to the maximum speed at which a FGTA approach can process incoming traffic data without dropping or losing packets. It is typically measured in terms of bits per second (bps) or packets per second (pps).

To increase efficiency, many FGTA approaches optimize the feature extraction procedure, classification algorithms, or the general pipeline. Some approaches also choose to design dedicated hardware architectures to accelerate the FGTA process. For example, FlowLens [58] utilizes programmable switch to support ML-based flow classification at hardware level, making ML-based flow classification efficient enough to catch up with line-speed traffic. We discuss more details about this issue in Section 2.4.

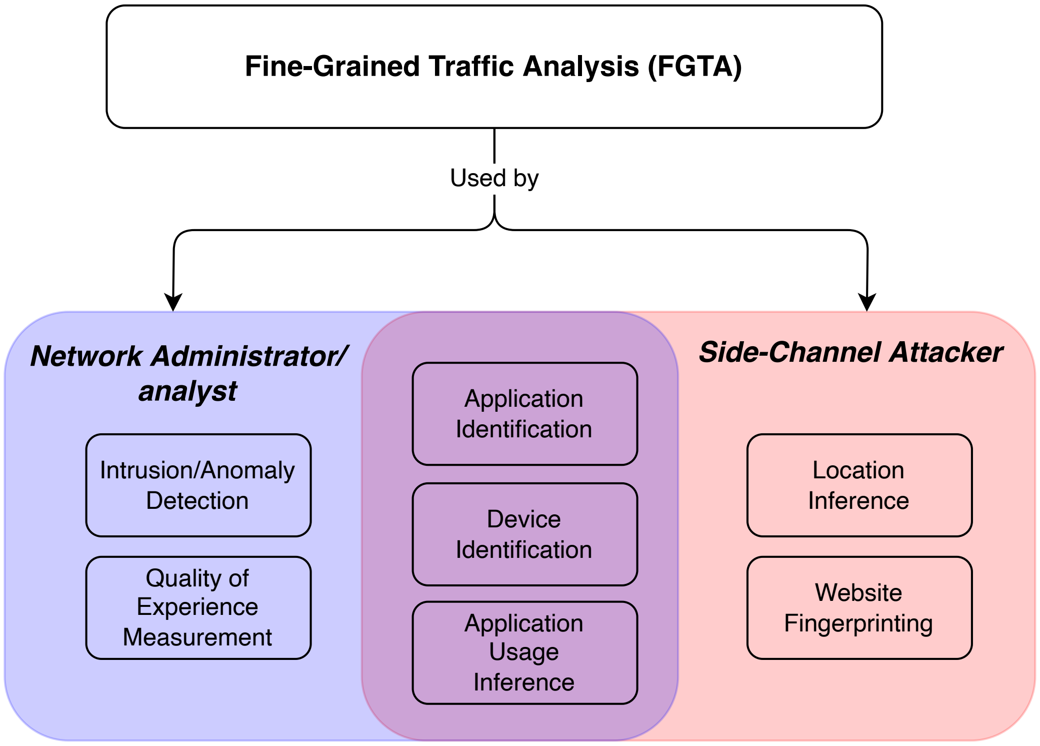

Fine-Grained Traffic Analysis (FGTA)
Used by
Network Administrator/ analyst
Intrusion/Anomaly Detection
Quality of Experience Measurement
Application Identification
Device Identification
Application Usage Inference
Side-Channel Attacker
Location Inference
Website Fingerprinting

Figure 8. A taxonomy for FGTA by use case.

2.3.4.3 ***Other metrics.*** According to different use cases, FGTA approaches may need to consider other metrics, such as the memory cost, the storage cost, compatibility, etc. Intrusion detection focused FGTA approaches may also need to consider the explainability of the outputs to help network administrators understand the reasons for the detection results, thereby facilitating the network security management with confidence. In addition, some FGTA approaches may need to consider the privacy of the users for compliance with certain regulations (i.e., GDPR). Thus, developers need to select suitable metrics for specific use cases.

2.4 Use Cases and Representative Approaches

As discussed in Section 2.1, FGTA has a wide range of uses. FGTA can be leveraged by both attackers and network administrators, for both illegal purposes and social good. According to their use cases, we propose a taxonomy for FGTA approaches (illustrated in Figure 8). In the rest of this section, We further examine typical FGTA approaches in each of the category.

2.4.1 Attack/Anomaly Detection.

TA approaches are widely used by both the industry and academia to detect anomalies or attacks. With more than two decades of research, we have seen a myriad of solutions (e.g.[143, 365, 149, 89]) targeting at different types of threats. However, as networks attacks become more and more sophisticated and traffic encryption is widely used by all the parties, detection approaches based on traditional TA gradually become incompetent to tackle modern attacks. Therefore, researchers began adopting FGTA to model hosts and clients' application-layer behaviors to detect such attacks/anomalies. In this subsection, we elaborate on FGTA-based attack/anomaly detection approaches, introducing their applicable scenarios and operation mechanisms (Table 5 shows an overview).

2.4.1.1 Intrusion detection.

Many FGTA approaches focus on detecting complicated intrusions in the network by examining the characteristics of the underlying network traffic. Most of them apply machine learning models to perform the detection.

Amoli et al. [48] leveraged an unsupervised machine learning model (i.e.density-based spatial clustering of applications with noise (DBSCAN)) to distinguish subtle differences between historic traffic and intrusion traffic. Their approach is able to detect zero-day and complex attacks without much prior knowledge of these attacks. Papadogiannaki et al. [298] generated traffic signatures from packet metadata sequences and then used these to detect intrusions in the UNSW-NB15 dataset [279].

Table 5. Comparisons of selected attack/anomaly detection approaches (○: not support; ◐: partially support; ●: support).

Category	Approach	Year	Goal of Analysis	Feature	Method	Real-World Evaluation
Intrusion Detection	Amoli et al. [48]	2016	Mail-bomb, SSH-process-table, botmaster, etc.	Flow-level traffic feature such as duration, number of packets, smallest packet size, largest packet size, etc.	DBSCAN	○
	Tang et al. [366]	2016	R2L, U2R, Probe, DoS	Duration, protocol type, src bytes, dst bytes, count, srv count	DNN	○
	Shone et al. [345]	2018	R2L, U2R, Probe, DoS, guess password, portsweep, buffer overflow, etc.	Features extracted with NDAE	NDAE for unsupervised feature learning, stacked NDAEs for detection	○
	Mirsky et al. [274]	2018	Video injection, ARP MitM, OS scan, etc.	Damped incremental statistics and 23 other features from packet-level data	Kitsune's core algorithm (KitNET), a type of autoencoders	●
Malware Detection	Shabtai et al. [340]	2014	Malicious attacks or masquerading/injected mobile applications	2 best feature subsets selected from 20 manually defined feature subsets of various sizes	Linear regression, decision table, SVM for regression, Gaussian processes for regression, isotonic regression, and decision/regression tree	◐
	Wang et al. [400]	2016	Android malware such as plankton, FakeInstall, FakeRun, MobileTx, etc.	Six TCP flow features and four HTTP request features	C4.5 decision tree	○
	Lashkari et al. [238]	2017	Malicious and masquerading applications such as Airpush, Kemoge, AVpass, FakeAV, etc.	24 features extracted from both packet and flow-level traffic	Random forest, KNN, decision tree, random tree, and regression	◐
	Anderson et al. [49]	2017	Detecting malicious, encrypted malware network traffic	22 and 319 data features in the standard and enhanced feature set extracted from NetFlow and IPFIX data	Linear regression, logistic regression, decision tree, random forest, SVM, and MLP	●
Data Exfiltration Detection	Ren et al. [320]	2016	Cross-platform information leak identification	Raw network packets with payload	Decision tree, AdaBoost, bagging, blending, and Naïve Bayes	●
	Continella et al. [110]	2017	PII leakage detection, even in the presence of obfuscation techniques	Raw network packets with payload	Behavior modeling and differential analysis	◐
	Rosner et al. [327]	2019	Information leaks in TLS-encrypted network traffic	A feature space that includes observations about individual packets and sequences of packets; additional features from the phase detection and the full original traces.	Trace alignment, phase detection, feature selection, feature probability distribution estimation and entropy computation	○
Others	Feng et al. [152]	2021	Online social network bot detection	Traffic fingerprint images converted from NetFlow data	DBSCAN, CNN	●
	Coulter et al. [112]	2019	A data-driven cyber security system that can identify high-level application-layer attacks or anomalies such as Twitter spam	Statistical features extracted from the network traffic and content (optional)	A varity of classification approaches	○
	Feng et al. [155]	2022	Cryptojacking activity	Packet size, timing, direction, and protocol from sFlow data	LSTM	○

Many researchers also focus on utilizing supervised deep learning models to detect intrusions. Tang et al. [366] extracted six basic features from traffic flows and trained a DNN model with the NSL-KDD dataset to detect intrusions. Shone et al. [345] first leveraged nonsymmetric deep autoencoder (NDAE) for unsupervised feature learning. Then, they implemented stacked NDAEs with GPU-based architectures for quick and accurate intrusion detection on labeled datasets (i.e.KDD Cup '99 and NSL-KDD). Mirsky et al. [274] monitored the statistical patterns of network traffic and designed an ensemble of neural networks called autoencoders to collectively differentiate between normal and abnormal traffic patterns. Their approach is able to detect various attacks (e.g.video injection, ARP MitM, OS scan, etc.). Besides, unlike many other approaches that are only evaluated in closed-world environments, this approaches was tested with a real-world test bed.

2.4.1.2 Malware detection. Today's malware is becoming more and more challenging to be detected by traditional TA due to traffic hiding and the increasing adoption of traffic encryption. FGTA could be an ideal tool to detect such malware.

Shabtai et al. [340] proposed a framework for malware detection on Android platforms. It can identify attacks or masquerading applications installed on a mobile device and injected applications with malicious code by semi-supervised machine-learning methods. Wang et al. [400] leveraged a machine learning algorithm (i.e.C4.5 decision tree) in analyzing mobile traffic, which is capable of identifying Android malware with high accuracy—more than 98%.

Later, some researchers collectively evaluated the efficacy of different machine learning models in detecting malware. Lashkari et al. [238] detected malicious and masquerading applications with five different classifiers—random forest, KNN, decision tree, random tree, and regression. They found that these models can achieve similar performances in malware detection. Besides, they published a labeled dataset that contains both benign Android applications and injected applications' network traffic. Anderson et al. [49] designed and carried out experiments that show how six machine learning algorithms (e.g.linear regression, logistic regression, decision tree, random forest, SVM, and MLP) perform when confronted with real network data. They found the random forest ensemble classifier to be the most robust for the domain of malware detection.

2.4.1.3 Data exfiltration detection. FGTA can also be used in detecting data exfiltration, thereby protecting personal sensitive data from leakage. Different from directly detecting anomalies or attacks, approaches in this domain usually profile user behaviors or model normal application usage to identify abnormal data transfer.

Wei et al. [411] proposed ProfileDroid, which is the first approach to profile mobile application at four layers: (a) static, or application specification, (b) user interaction, (c) operating system, and (d) network. At network-layer, this approach can capture essential characteristics of application communications, including but not limited to the ratio of incoming traffic and outgoing traffic, number of distinct traffic sources, traffic intensity, the percentage of HTTP and HTTPS traffic, etc. The profiling information can help identify inconsistencies and surprising behaviors, thereby detecting data exfiltration. A similar work is TaintDroid [139]. It leverages dynamic information-flow tracking to identify private data leaks of Android applications. The authors indicated that network traffic is useful to help monitor the behavior of popular third-party Android applications and discover potential misuse cases of user private information across applications. These two approaches do not only leverage network traffic, but their ideas inspired a lot of subsequent work in this domain.

Later, researchers began to investigate purely using network traffic to profile application usage and report possible data exfiltration. Razaghpanah et al. [318] monitored network communications on mobile phones from user-space. The proposed approach facilitates user-friendly, large-scale deployment of mobile traffic measurements and services to illuminate mobile application performance, privacy and security. Song et al. [356] proposed a VPN-based approach to detect sensitive information leakage Le et al. [240] proposed AntMonitor, which passively monitors and collects packet-level measurements from Android devices to provide a fine-grained analysis. By inspecting the network traffic data, it can provide users with control over how their data is shared by applications. Ren et al. [320] proposed ReCon, a cross-platform system that reveals personally identifiable information (PII) leaks by inspecting network packets and gives users control over them without requiring any special privileges or custom operating system (OS). The authors leveraged the Weka data mining tool [186] to train classifiers that predict PII leaks. Continella et al. [110] proposed an approach to privacy leak detection that is resilient to obfuscation techniques (e.g.encoding, formatting, encryption). To achieve the goal, the authors first established a baseline of the network behavior of applications, and then utilized black-box differential analysis on application usages.

However, the aforementioned approaches still require inspections on traffic content to detect data exfiltration. The ideal FGTA-based solution should be content-agnostic. In 2019, Rosner et al. [327] presented a black-box approach for detecting and quantifying side-channel information leaks in TLS-encrypted network traffic. Given a user-supplied profiling-input suite in which some aspect of the inputs is marked as secret, it combines network trace alignment, phase detection, feature selection, feature probability distribution estimation and entropy computation to quantify the amount of information leakage that is due to network traffic.

2.4.1.4 Others. A few research works have been focusing on using FGTA to detect other types of application-layer anomalies. By harnessing the power of machine learning on big data, such approaches can model fine-grained application-layer anomalies only with flow-level traffic or packet headers. For example, BotFlowMon [151, 152] detects online social network bot traffic by converting NetFlow records to images and training a convolutional neural network (CNN)-based classification model; Coulter et al. [112] proposed a data-driven cyber security system that can detect Twitter spam or other high-level application-layer anomalies through machine-learning-based flow analysis; Feng et al. [157, 155] detects cryptojacking traffic by inferring the hash rate stability with cryptomining traffic in sFlow format. Table 5 lists their analysis features and methodologies.

2.4.2 Fine-Grained Quality of Experience Investigation.

Quality of experience (QoE) is a well-studied topic in the development of the Internet. Unlike quality of service (QoS), which refers to the network parameter settings configured by service providers to deliver various levels of service to their customers, QoE measures how the service is experienced by individual users at the edge of the network [227]. To score well in QoE, service providers need to analyze network traffic to conduct QoE investigations. The investigation results can help them revise the network configurations accordingly, thereby providing decent service to users.

Table 6. Comparisons of selected fine-grained QoE investigation approaches.

Approach	**Goal**	**Method**	**Feature**
[129]	Identify QoE degradation in YouTube	Random forest	Three feature sets selected by information gain.
[293]	Estimate QoE in YouTube	Random forest, J48, Naïve Bayes, OneR, and SMO	Five hand-crafted feature sets
[265]	Estimate video streaming QoE over HTTPS and QUIC protocols	Decision tree	A packet-level feature set extracted from network and transport-layers
[228]	Estimate QoE in YouTube	Random forest and linear regression	Three feature sets (inbound, outbound, and inbound + outbound)
[426]	Estimate mobile ABR video adaptation behavior over HTTPS and QUIC protocols	Traffic fingerprinting with chunk sizes	Packet size and timing

Plenty of works have been proposed to conduct QoE investigations with traditional TA (e.g.[227, 206, 39]). They can roughly classify network traffic into several groups (e.g.video, voice, data transfer, etc.) using statistical, DPI-based, or rule-based approaches and measure the service experience according to some metrics. However, such approaches may not be able to tackle today's increasingly complicated network traffic, since different types of traffic may be encrypted by different protocols (e.g.HTTPS, Quick UDP Internet Connection (QUIC)) and sent from different devices (e.g.Internet of things (IoT), smartphone, server) by different applications. Besides, service providers may want to conduct more granular management of network traffic. For example, residential areas' network administrators may want to increase the priority of video streaming traffic related to YouTube for certain users; network administrators of companies may want to ensure the quality of online meeting traffic for some offices. Therefore, researchers began to leverage FGTA to conduct QoE investigation in finer granularities in the past ten years.

Usually, fine-grained QoE investigations are performed in two steps:

1. Extract the target traffic using traffic classification.

2. Measure the extracted traffic to check if it meets certain criteria.

Some approaches may combine these two steps into one and directly identify potential QoS/QoE problems. Table 6 shows a comparison of some selected QoE methods.

In 2016, Dimopoulos et al. [129] proposed a random-forest-based detection model to identify QoE issues related to YouTube video streaming. By selecting three sets of features with information gain, the proposed model is able to directly detect different levels of QoE degradation that is caused by three key influence factors (i.e.stalling, the average video quality, and the quality variations). The authors demonstrated that it can detect QoE problems with an accuracy of 92% by evaluating this approach using collected traffic At the same year, Orsolic et al. [293] also studied using different machine learning algorithms (i.e.random forest, J48, Naïve Bayes, OneR, and Sequential Minimal Optimization (SMO)) to detect YouTube QoE issues under different bandwidth scenarios. In 2019, Khokhar et al. [228] proposed the first work that not only can identify YouTube QoE issues related to objective factors (e.g.startup delay, stalling, resolution change, etc.), but also can identify QoE issues related to the subjective Mean Opinion Score (MOS).

Mazhar et al. [265] further extends QoE investigation to all encrypted video streaming traffic (transferred over HTTPS or QUIC) by using a classification model trained by decision tree. They demonstrated that their approach is able to achieve a 90% classification accuracy for HTTPS and an 85% classification accuracy for QUIC Xu et al. [426] infers mobile Adaptive Bitrate (ABR) video adaptation behavior using packet size and timing information in encrypted environments.

2.4.3 Website Fingerprinting.

Website fingerprinting (WFP) is used to identify what web page the user is visiting, even in the presence of traffic encryption or encrypted tunnels established by Tor [266, 304], Shadowsocks (i.e.a popular secure socks5 proxy) [9], VPN, etc. It is a FGTA technique that widely-used by attackers to eavesdrop on user activities online. In this subsection, we survey and compare well-known WFP approaches (Table 7), elaborating on the history of WFP and investigating its capability.

2.4.3.1 Early development of WFP.

WFP has a long history. The early WFP attacks simply focused on using data sizes to infer the URL the user is visiting through encrypted SSL connections. Back in 1998, Mistry et al. [275] demonstrated that the size of HTML files is a critical feature to specific web pages. They proposed an attack that simply uses the transmitted data volumes to identify certain websites. Although this attack is not feasible anymore after the launch of connection pipelining and connection parallelization by HTTP 1.1 (RFC 2616 [161]), this research inspired many other WFP researches in the next two decades. In 2002, Hintz [198] defined "fingerprints" of websites as the histograms of transferred files' sizes. He recorded some website fingerprints and successfully recognize some websites transferred through HTTPS with these fingerprints. However, Hintz's WFP attack only works for a small number of websites. Later, Sun et al. [359] extends size-based WFP to thousands of websites. They proposed a WFP approach based on Jaccard's coefficient, which can correctly identify 75% of the websites in their collected dataset. However, a common drawback of file-based attacks is that they cannot tackle traffic hidden in encrypted tunneling protocols (e.g.VPN, OpenSSH), not to mention Tor.

Table 7. Comparisons of selected WFP approaches (○: not support; ◐: partially support; ●: support).

Approach	Year	Method	Feature	Effectiveness			
				HTTP/1.1	VPN	Tor	Multi-tab
Mistry et al. [275]	1998	Size matching	Size of HTML file	○	○	○	○
Sun et al. [359]	2002	Similarity score calculation (Jaccard's coefficient)	HTTP object count, sizes, etc.	◐	○	○	○
Bissias et al. [74]	2005	Cross correlation of two value sequences	Packet size and inter-arrival time distributions	●	◐	○	○
Liberatore et al. [252]	2006	Similarity score calculation (Jaccard's coefficient)	Direction and length for each packet	●	◐	○	○
Herrmann et al. [197]	2009	Multinomial Naïve Bayes	Frequency distribution of the IP packet size	●	●	○	○
Panchenko et al. [296]	2011	SVM	Volume, time, and direction of the traffic	●	●	◐	○
Cai et al. [82]	2012	Damerau-Levenshtein distance and Hidden Markov Model	Packet size, time, and direction	●	●	●	○
Wang et al. [402]	2014	KNN	A large feature set generated from packet-level traffic	●	●	●	○
Hayes et al. [192]	2016	Random decision forests	Features selected by gini coefficient	●	●	●	◐
Rimmer et al. [323]	2017	SDAE, CNN, and LSTM	Automatically learned feature sets from packet-level network traffic	●	●	●	○
Sirinam et al. [348]	2018	CNN	Packet-level traffic data	●	●	●	○
Sirinam et al. [349]	2019	N-shot learning with triplet networks	Selected by a neural-network-based feature selector	●	●	●	○
Yin et al. [433]	2021	Split point finding and BalanceCascade-XGBoost	Packet size, time, and direction	●	●	●	●

2.4.3.2 Defeat encrypted tunnel. To extend WFP to handle encrypted tunneling protocols, multiple "more advanced" WFP approaches had been proposed. Both Bissias et al. [74] and Liberatore et al. [252] proposed improved forms of WFP. Rather than using the data size as the feature, they extract sets of traffic patterns from encrypted IP packet headers, such as packet inter-arrival time, size, etc. These approaches have some efficacy in identifying websites transferred by encrypted tunneling services. However, the accuracies of page identification is still not ideal in reality. In 2009, by using packet-level features, Herrmann et al. [197] proposed a multinomial Naïve Bayes classifier that can identify up to 97% of web requests on a sample of 775 sites and over 300,000 real-world traffic dumps recorded over a two-month period. The authors demonstrate that this approach is effective in tackling website traffic in encrypted tunnels. Lu et al. [261] pointed out that packet ordering information, though noisy, can be utilized to enhance website fingerprinting. In addition, the ordering information is effective for WFP even under traffic morphing. By calculating the Levenshtein distance between different network traffic, their approach can perform WFP over OpenSSH for 2,000 profiled websites. The identification accuracy of the proposed scheme reaches 81%, which is 11% better than the approach proposed by Liberatore et al. [252].

2.4.3.3 WFP in Tor era. To safeguard personal information and avoid Internet censorship in an increasingly dangerous network environment, many people began to use The Onion Router (Tor), a free and open-source software for enabling anonymous communication, to visit the Internet. Different from traditional encrypted tunneling protocols, Tor reroutes Internet traffic through a worldwide, volunteer overlay network, consisting of more than six thousand

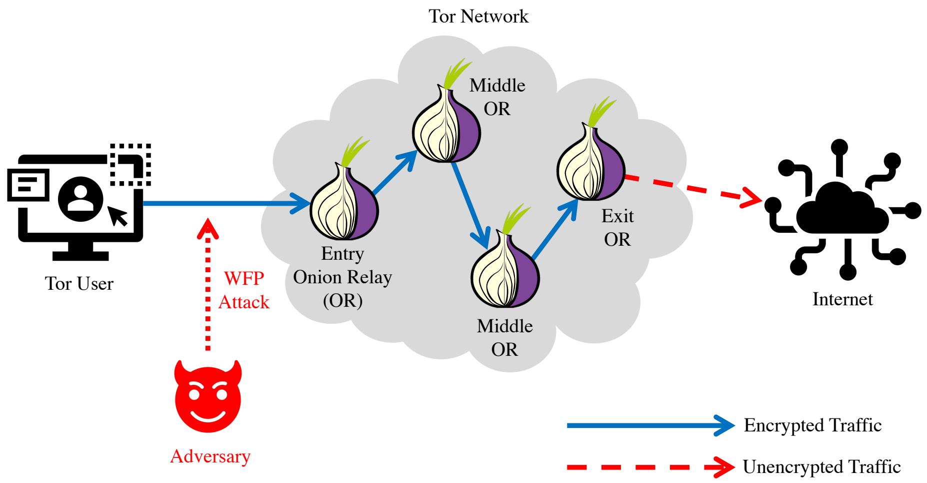


Figure 9. Threat model for WFP attacks over Tor network.

relays [282], for concealing a user's actual location and Internet usage from anyone conducting network surveillance or TA.

Figure 9 illustrates the operation model of Tor. To protect user's identity, each Tor user creates an encrypted virtual tunnel to its destination through a chain of several volunteer nodes—onion relays (ORs). According to their positions in the virtual tunnel, ORs can be classified into entry OR, middle OR, and exit OR. Each of the ORs only knows its predecessor and its successor [122]. When forwarding network traffic, the user's network packets will be encrypted in multiple layers and each of the ORs can only decrypt one layer of encryption. Thus, Tor ensures that none of the ORs in the circuit knows the user and its destination at the same time. Besides, to prevent TA, the user data is encapsulated in chunks of a fixed size, called cells, before transmission [130]. The WFP attacks above are thus ineffective against Tor network, as they rely heavily on packet-size-related features.

Indeed, it is almost impossible to find any useful knowledge inside a Tor network. However, the virtual tunnel between the Tor user and the entry OR does provide attackers with an interface and makes WFP possible (illustrated in Figure 9).

In 2011, Panchenko et al. [296] are the first to demonstrate that it feasible to use WFP to identify web pages visited by Tor users. They trained a SVM classifier with features extracted from volume, time, and direction of network packets, with a classification accuracy of 55% when testing with their web page dataset. Panchenko et al. are also the first to evaluate their WFP attack in a real-world setting. The result shows that their approach is able to achieve a true positive rate of up to 73% and a false positive rate of 0.05%. Based on this work, a significant amount of improved WFP approaches were proposed to use different algorithms and features (e.g.VNG++ [136], Hidden Markov Models [82], Levenshtein-like distance [403], etc.) to tackle web page identification in Tor. In 2014, Wang et al. [402] proposed a KNN WFP classifier and applied it on a large feature set with weight adjustment. Their approach achieved an accuracy of 91% in a closed-world setting and a true positive rate of 85% for a false positive rate of 0.6% when testing with more than 5,000 background pages in a real-world setting.

Nevertheless, these WFP approaches still have some obvious flaws according to an evaluation made by Juarez et al. [221]:

- Previous WFP attacks assume single-tab browsing behavior of users. However, multi-tab browsing is widely used in reality.

- WFP attacks highly depend on the coverage of training dataset, but existing datasets cannot include web page traffic from all versions of Tor browser, user habits, or user locations.

– Previous WFP attacks cannot detect dynamic or personalized web pages, as they traffic of these pages is polytropic.

– Many countermeasures for WFP have been proposed (which will be discussed later in Section 2.6), making many of previous WFP attacks non-effective.

To further increase the success rate of WFP attacks and defeat countermeasures, researchers began to collect more comprehensive training datasets, use more complicated feature sets, and apply more sophisticated classification algorithms for WFP.

Wang et al. [403] described how they collect the training dataset in a much more thorough manner than previous works. They gathered the data in different Tor settings and with different defense approaches. Later, Panchenko et al.[295] collected the first Internet-scale WFP dataset to develop and evaluate WFP comprehensively. Based on the dataset, they proposed CUMUL, a web page classifier that has a higher recognition rate and a smaller computational overhead than previous approaches. They also demonstrated that although CUMUL is more efficient and superior in terms of detection accuracy, still, it cannot scale when applied in realistic settings. As for WFP feature set, Cai et al. [81] systematically analyzed previous WFP approaches to understand which traffic features convey the most information; Hayes et al. [192] utilized the gini coefficient index to select a feature set and designed a random decision forests classifier based upon them; Wang et al. [404] evaluated the classification accuracy of each feature category by using KNN.

In the recent five years, the development of WFP has been focusing on conducting attacks in the presence of effective countermeasures, with little encrypted data, or under complicated circumstances. Many of recent approaches also investigated the applicability of deep learning techniques in WFP. Rimmer et al. [323] trained three classification model with Stacked Denoising Autoencoder (SDAE), CNN, and Long Short-Term Memory (LSTM) respectively. These deep learning models are capable of automatically learning the best features to conduct WFP. The authors further demonstrated that automatically-created features are more effective especially in tackling constantly changing web content. In 2018, Sirinam et al. [348] presents a very powerful WFP attack—Deep Fingerprinting (DF). By employing a CNN model with a sophisticated architecture design, the authors claim that this attack can defeat many WFP countermeasures (e.g.WTF-PAD [222] and Walkie-Talkie [404]) and works well in very complicated real-world scenarios (95% accuracy for 20,000 URLs in a real-world setting). Sirinam et al. [349] further proposed an approach based on N-shot learning with triplet networks in 2019, which can achieve decent efficacy with relatively small training data. Besides these approaches, Abe et al. [35] also applied SDAE in WFP; Bhat et al. [66] leveraged ResNets [193], a CNN architecture, to reach high success rates in WFP; Oh et al. [290] used unsupervised DNN to generate low-dimensional features and trained different machine learning classification models based upon them. In 2021, Wang et al. [391] leveraged adversarial domain adaption (a transfer learning technique) to achieve high WFP accuracy with little encrypted data; Yin et al. [433] proposed a WFP attack that is able to identify websites in multi-tab environments, which means it can achieve usable accuracies regardless of the number of simultaneously opened web pages; Hoang et al. [199] found that even in the presence of domain name encryption technologies or content delivery network (CDN), WFP based on IP addresses is still feasible. They exploited the complex structure of most websites, which load resources from several domains besides their primary one, and further applied the generated domain fingerprints to conduct WFP at large.

2.4.4 Location Inference. Location inference is a widely studied topic by computer scientists. We have seen myriad works focusing on using social network information [41, 211], smartphone accelerometer [187], image content [168], etc. to infer users' locations. In the past decade, a few researchers began to use FGTA to conduct location inference. The location we discuss here can be either a geographical location or a contextual location, the later one means the type of location the user is sending packets from, such as an airport, a campus, or a residential building. This subsection examines inference approaches for these two types of locations.

2.4.4.1 Contextual location inference. The intuition behind contextual location inference with FGTA is straightforward—users from different types of locations tend to generate different traffic because they need to use different web applications at different locations. Besides, different locations (e.g.campus, company, residential area) may process network traffic in different manners. Contextual location inference using FGTA aims to measure and analyze sets of network traffic and infer where these sets of traffic are coming from.

Back in 2009, Trestian et al. [376] conducted a detailed study on applications accessed by users at different locations. They demonstrated that users are more likely to show interest in a particular class of applications than others at certain locations, which is irrespective of the time of day. They indicated that we can further use the traffic generated by these applications to identify the type of locations (e.g.work versus home). In 2014, Das et al. [117, 118] collected around 100 GBs of real-world network traffic from more 1700 users at different types of

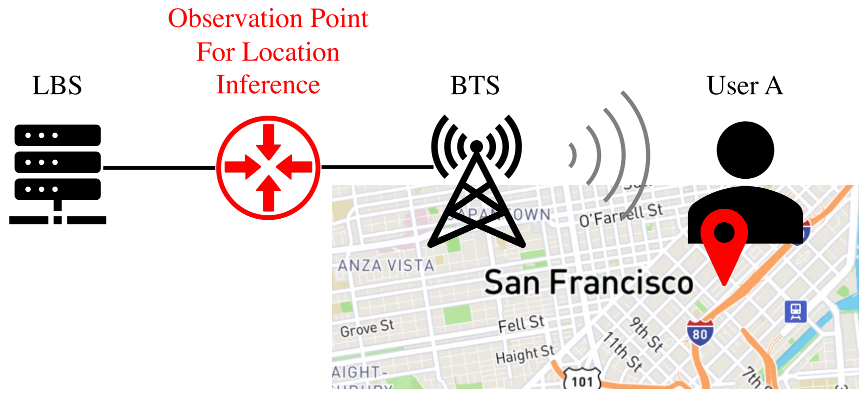


Figure 10. Operation model for geographical location inference.

locations (e.g.cafeteria/restaurant, university campus, airport/travel, etc.). By measuring and analyzing this dataset, Das et al. selected sets of features for packet-level, flow-level traffic and built a decision-tree-based classification model to predict contextual location with an overall accuracy of 87%. Later, a few similar works also demonstrated that mobile traffic from different cellular towers [423, 395, 424] tends to have different characteristics.

The drawback of contextual location inference is that it only works on a group of network traffic sending from many endpoints. It cannot infer a device's contextual location by only analyzing its own network traffic.

2.4.4.2 Geographical location inference. Purely using network traffic to infer a user's geographical location seems impossible. However, in 2015, Ateniese et al. [52] demonstrated that it is actually feasible under certain assumptions.

Nowadays, location-based applications (LBA), such as Facebook, Yelp, Google Map, etc., are widely used. These LBAs obtain user locations through location-based services (LBS). LBS providers usually use a base transceiver station (BTS) to locate a user and send real-time location information to the user. Ateniese et al. proposed an approach (illustrated in Figure 10) that simply monitors the traffic between the BTS and the LBS to identify user locations. They found that different locations will trigger LBS packets of different sizes. An adversary can potentially create a location knowledge base of different locations' packet sizes and their corresponding timestamps to conduct geographical location inference. Still, this approach has many limitations (e.g.low accuracy, difficult to build the location knowledge base at large, etc.). This work is more about demonstrating the feasibility of geographical location inference with FGTA than launching a full-fledged, ready-to-use approach.

2.4.5 Device/OS Identification. TA has been used to identify user devices or the OS running on the device for a long time. For example, Lippmann et al. [254] focused on extracting TCP or IP packet metadata to recognize different OSes in 2003. However, with the increase in the variety and complexity of user device and OS, simply identifying the device/OS type according to rules in packet header is no longer effective. Thus, researchers turn to use FGTA to investigate if specific traffic patterns can be correlated with some OSes or devices, which not only can recognize a few rough device/OS types, but also can pinpoint the device model or OS for various IoT and mobile devices. In this subsection, we introduce such FGTA approaches that deal with device/OS identification.

2.4.5.1 OS identification. Chen et al. [93] perform OS identification and detection of NAT and tethering (i.e.multiple devices sharing the Internet connection of a mobile device, which can lead to multiple OSes sharing a single IP address) by inspecting TCP/IP headers of packet traffic. They leverage a probability-based method by applying the Naïve Bayes classifier to effectively combine multiple features (e.g.TTL value, IP ID monotonicity, TCP timestamp, clock frequency, etc.), thereby fingerprinting and recognizing different OSes in different environments. Laštovička et al. [239] also proposed an OS identification method by constructing a decision tree with the TLS handshake, HTTP headers, and TCP/IP features. However, these approaches cannot distinguish between minor versions of the same OS. To tackle this problem, Ruffing et al. [332] identify different versions of smartphone OSes by using the frequency spectrum of packet timing from encrypted traffic. By identification through correlations of the feature-extracted spectra, the authors demonstrate that even a network traffic input of 30 seconds can be enough for high-accuracy identification results.

2.4.5.2 IoT device identification. Compared with OS identification, IoT device identification can be more challenging due to the complexity of their network environments and the devices' wide variety. Lopez-Martin et al. [257] extract a time-series feature vectors from network traffic, where each element of the time-series vector contains the features of a packet in the flow. They then proposed a classifier that is based on both a RNN model and a CNN model to separate heterogeneous IoT traffic using the features. Meidan et al. [269] collected and labeled network traffic from nine distinct IoT devices (e.g.baby monitor, motion sensor, printer, security camera, etc.), PCs, and smartphones. They then utilized a multi-stage machine-learning-based classifier to classify traffic of IoT devices in two phases. In the first stage, the classifier can distinguish traffic between IoT and non-IoT devices. In the second stage, the classifier can further identify traffic from different IoT devices. The authors demonstrate that their approach is able to classify IoT traffic with an accuracy of 99.281%.

However, these two researches do not consider complicated network environments (e.g.smart homes, enterprises, and cities) of IoT devices. Sivanathan et al. [351] addressed this challenge by developing a robust framework for IoT device traffic classification with a multi-stage machine-learning-based algorithm. The authors instrumented a smart environment with 28 different IoT devices that consist of spanning cameras, lights, plugs, motion sensors, appliances, and health-monitors. They then collected and synthesized network traffic traces from this infrastructure for a period of six months. By extracting statistical features such as activity cycles, port numbers, signaling patterns, and cipher suites from the traffic and using Naïve Bayes and random forest as the identification models, they are able to classify heterogeneous IoT devices with an accuracy over 99%. Yao et al. [430] further proposed an end-to-end IoT traffic classification method that eliminates the multi-stage classification for high accuracy and efficiency. It relies on a deep-learning-aided capsule network to construct an efficient classification mechanism that integrates feature extraction, feature selection, and classification model. One drawback of these approaches is that their evaluations are all based on closed-world datasets, which may not be able to precisely reflect their true efficacy in the real world.

2.4.6 Application Identification. Using the network traffic from a device to identify the applications that are running on the device, even in the presence of traffic encryption, is one of the most classic use cases of TA. Decades ago, people have investigated using traditional TA approaches to classify traffic from different applications. Before 2000, many researchers simply used traffic ports to identify some popular applications that have well-established ports (e.g., port 443 for HTTPS, port 110 for POP3). Port-based approaches fail for most emerging applications such as gaming, streaming, and messaging [63]. Later, Karagiannis et al. proposed BLINC [225], which not only looks at port-based features, but also inspects the host's social behavior and its community behavior to determine the applications. Bernaille et al. [62] observe the sizes of the first few packets of an SSL connection to identify the web application, which can achieve an accuracy of more than 85%. There are also many machine-learning-based traditional TA approaches [64, 267, 140, 277] that classify application traffic according to the traffic patterns.

However, application identification with traditional TA can hardly adapt to the current network environment and meet current needs due to several limitations:

- Traditional TA can only identify some high-level protocols (e.g.HTTP, HTTPS, SMTP, POP3, etc.) and a few frequently used applications that have obvious traffic patterns (e.g.MySQL, BitTorrent, MSN, etc.).
- Traditional TA-based application identifications only work in relatively simple network environments. For example, endpoints only consist of servers, clients, and peers; devices communicate without encrypted tunneling protocols (e.g.virtual private network (VPN)).

Nowadays, network environments are becoming far more complicated that before. Different types of nodes (e.g.smartphone, IoT, middlebox) may communicate through complicated network environments (e.g.VPN, network address translation (NAT), WiFi). Besides, millions of web applications are used on different platforms, with more complex communication mechanisms and much less regular traffic patterns. Therefore, people started to leverage FGTA in identifying specific applications among miscellaneous traffic from different types of devices. In this subsection, we introduce typical FGTA-based application identification approaches (Table 8 shows an overview).

Table 8. Comparisons of selected FGTA approaches for application identification (○: not support; ◐: partially support; ●: support).

Category	Approach	Year	Target Application	Traffic Feature	Method	Real-World Evaluation
App Identification for General-Purpose Devices	Chen et al. [94]	2017	Instagram, Skype, Facebook, Wechat, Youtube, etc.	Images converted from flow sequences	RKHS-based data fusion and CNN	○
	Rezaei et al. [321]	2018	Google Drive, Youtube, Google Docs, Google Search, Google Music, etc.	Time series features extracted from sampled packets	Semi-supervised CNN	○
	Lotfollahi et al. [258]	2020	Vimeo, YouTube, VoipBuster, Spotify, Netflix, Hangouts, Facebook, etc. (with or without VPN)	Normalized features extracted from packet headers	CNN and SAE	○
Mobile App Identification	Wang et al. [399]	2015	Snapchat, Tecent QQ, Mint, Tinder, YouTube, etc.	Statistical features from packets (e.g., STD time, average size, STD size, etc.)	Random forest	◐
	Alan et al. [44]	2016	1595 applications on four different devices	Features from TCP/IP headers (e.g., packet size, timing, direction)	Jaccard's coefficient and Naïve Bayes	◐
	Taylor et al. [367]	2016	110 most popular applications in Google Play Store	Two sets of features from flow-level traffic-flow vector and statistical features	SVM and random forest	◐
	Aceto et al. [36]	2018	49 mobile applications (i.e., QQ, SayHi, eBay, 6Rooms, NetTalk, PureVPN, etc.)	Statistical features (e.g., packet length, percentiles, deviation, etc.) for incoming, outgoing, and bidirectional packets	A multi-classification (viz. fusion) model consists of Naïve Bayes, random forest, SVM, and decision tree	○
	Aceto et al. [37]	2019	Facebook, Facebook Messenger, and other 49 apps on both Android and IOS	Automatically-extracted features using neural networks	Multiple machine learning models (e.g., CNN, LSTM, MLP, etc.)	○
	Van et al. [380]	2020	More than 1M apps from three datasets.	Packet and flow-level features selected by adjusted mutual information	A semi-supervised fingerprinting with destination-based clustering, browser isolation, and pattern recognition	○
Decentralized App Identification	Shen et al. [343]	2019	Aragon, Bancor, Canwork, Chainy, Cryptopepes, Eth_town, Etheremon, etc.	57 features of packet lengths, 72 features of bursts, and 54 features of time series, fused by kernel functions	KNN, SVM, and random forest	◐
	Aiolli et al. [40]	2019	BTC.com. BitPay, Bread, Wirex, Copay, etc.	Vectors of statistical features about the packet length from traffic flow	SVM and random forest	◐

2.4.6.1 Application identification for general-purpose devices.

General-purpose devices, such as personal computers and servers, support the operation of countless web applications. Recently, FGTA-based application identification for general-purpose devices focuses on identifying more specific applications in more complicated network environments.

Chen et al. proposed Seq2img [94], an application traffic classification framework based on an online CNN model. Seq2img employs a data fusion method based on Reproducing Kernel Hilbert Space (RKHS) to convert flow sequences into images, which can fully capture the static and dynamic behaviors of different applications. Then, Seq2img utilizes a CNN model to recognize network traffic of popular applications, such as Facebook, Instagram, Wechat, etc.

Rezaei et al. [321] investigated using a few labeled, sampled packet-level datasets to train a comprehensive application identification model. They first pre-train a CNN-based model on a large unlabeled dataset, where the input is the time series features of a few sampled packets. Then, the learned weights are transferred to a new CNN model that is re-trained on a small labeled dataset. They demonstrated that this semi-supervised approach achieves almost the same accuracy as a fully-supervised method with a large labeled dataset. The proposed approach is able to identify applications like Google Drive, Google Doc, Google Search, Google Music, etc.

In 2020, Lotfollahi et al. [258] proposed an application identification method that can work in both VPN and non-VPN networks. After extracting features from packet headers, they used both CNN and stacked autoencoder (SAE) to train the classification models. Evaluation results show that this approach can achieve a recall score of 0.98 in application identification tasks.

2.4.6.2 Mobile application identification. With the raising of mobile network, mobile application identification becomes an emerging research topic in recent years. Unlike general-purpose devices, mobile devices are less regularized in port usage. In addition, a wide variety of mobile applications may utilize some common libraries in communication, generating similar network traffic patterns. Thus, mobile application identification can be more challenging.

Wang et al. [399] use random forest algorithm to analyze packet-level traffic in wireless networks. Their approach is able to detect the usage of 13 selected popular mobile applications on IOS platform, such as Snapchat, Tecent QQ, Mint, Tinder, YouTube, etc., with an accuracy of more than 87.23%. They demonstrate that by using the mobile applications the privacy of the user is more at risk compared to using online services through browsers on mobile devices.

Many researchers also studied application identification on Android platform. Inspired by some WFP approaches (Section 2.4.3), Alan et al. [44] use Jaccard's coefficient and Naïve Bayes to analyze features (e.g.packet size, timing, direction) from TCP/IP headers to identify 1595 applications on four different devices. Taylor et al. [367] proposed AppScanner, a framework that can automatically fingerprint and identify Android applications from their encrypted network traffic. The authors extracted two sets of features (i.e.flow vector and statistical features) from flow-level network traffic and implemented this approach using both SVM and random forest algorithms. The evaluations show that AppScanner can identify the 110 most popular applications in Google Play Store with more than 99% accuracy. In the next year, Taylor et al. further extended AppScanner in a follow-up research [368]. They investigated how application fingerprints change over time, across different devices, and across different application versions.

Recently, many similar works (e.g.[36, 37, 380, 38]) have been proposed to enhance the efficacy, efficiency, and coverage of mobile application identifications.

2.4.6.3 Application identification on other platforms. A few researches have been focusing on identifying decentralized applications on blockchain systems. Shen et al. [343] proposed an encrypted traffic classification of decentralized applications (e.g.Cryptopepes, Matchpool, Lordless, etc.) on Ethereum with features like packet lengths, bursts, and time series. Aiolli et al. [40] focused on identifying user activities on Bitcoin wallet applications (e.g.BTC.com, Bitcoin Wallet, Coinbase, etc.). The authors used SVM and random forest models to conduct the identification.

We also studied the application identification approaches for IoT devices. However, as each IoT device is usually bundled with a IoT application, the identification of IoT application is equal to the identification of IoT devices in most cases. Therefore, we introduce these approaches in Section 2.4.5 (IoT device identification).

2.4.7 Application Usage Inference. Application usage inference aims to analyze encrypted network traffic to identify certain application events, infer user behaviors, and measure specific service usage. It is one of the most challenging FGTA tasks, as it not only classifies the network traffic that is associated with different applications, device, or web pages, but also leverages the traffic patterns to recognize the application-layer activities that users conducted with the applications, devices, or web pages. Therefore, many application usage inference approaches may take extra steps (e.g.clustering, pre-filtering, etc.) to

Table 9. Comparisons of selected application usage inference approaches (○: not support; ◐: partially support; ●: support).

Category	Approach	Year	Analysis Object	Feature	Method	Real-World Evaluation
Messager/OSN	Coull et al. [111]	2014	Apple iMessage: language, control, read, start, stop, image, text, etc.	Payload length and the message length; a binary feature vector of packet length and direction pairs	Linear regression, Naïve Bayes, and rule lookup table	○
	Fu et al. [166]	2016	Wechat and WhatsApp: stream video call, news feed, location sharing, etc.	Discriminative features from the perspectives of packet length and time delay	Traffic segmentation with hierarchical clustering and thresholdingheuristics; HMM-based classifier.	◐
	Liu et al. [255]	2017	Facebook, Wechat, and WhatsApp: short video, video call, text, picture, etc.	A selected feature set extracted from traffic packet sequences by a Maximizing Inner activity similarity and Minimizing Different activity similarity measurements.	A recursive time continuity constrained K-means clustering algorithm for traffic flow segmentation and a random forest classifier for segmented traffic classification.	●
	Feng et al. [152]	2021	Facebook and Twitter: post, chat, read, etc.	Images converted from NetFlow records	Clustering-based traffic segmentation; CNN	●
Streaming Service	Wright et al. [417]	2008	Identify the phrases spoken within a call from a standard speech corpus.	The lengths of encrypted VoIP packets	HMM	○
	Schuster et al. [337]	2017	Identify the videos streamed by YouTube, Netflix, Amazon, and Vimeo.	Time series data of the following flow attributes: down/up/all bytes per second, down/up/all packet per second, and down/up/all average packet length.	Time-based burst; CNN	○
General-Purpose	Conti et al. [108]	2015	User activities in Gmail, Facebook, Twitter, Tumblr, Dropbox, etc.	Features from TCP/IP packet fields (e.g., IP address, port number, packet size, direction, and timing)	Dynamic time warping, random forest, and a hierarchical clustering algorithm called agglomerative	○
	Saltaformaggio et al. [333]	2016	User activities on Android and IOS platforms	Features extracted from IP packet headers, divided by behavior measurements (a small time window)	A K-means clustering model and an SVM model	◐
	Papadogiannaki et al. [297]	2018	User activities (e.g., voice call, video call, messaging, etc.) in popular Over-The-Top mobile applications (e.g., WhatsApp, Skype, Viber, etc.)	Customizable	A pattern language to identify application events, rule mining	●
Others	Yan et al. [428]	2018	Red packet transactions and fund transfers in Wechat	Overall statistics, packet length, number of TCP handshakes, inbound and outbound statistics	Threshold-based traffic segmentation, random forest	○
	Wang et al. [409]	2019	Classify specific actions (e.g., transfer payment, transfer receipt, QR code payment, etc.) on the mobile payment application, and then detect the detailed steps (e.g., click the button, receive the fund, open the red packet, etc.) within the action	Overall statistics of the packet length, range statistics of the packet length, flow statistics, incoming and outgoing statistics.	Threshold-based traffic segmentation, hierarchical identification with random forest, AdaBoost, GBDT, and XGBoost	○
	Jiang et al. [218]	2019	Application usage information (e.g., reading documents, surfing webs, editing documents, etc.) on remote desktop	Statistic features of flow burst	Threshold-based traffic segmentation, logistic regression, SVM, GBDT, random forest	○
	Wang et al. [408]	2020	Identify DApp (e.g., Superrare, Editional, John Orion Young, etc.) user behaviors (e.g., open DApps, open market, view detail, etc.)	Selected DApps features, behavior-sensitive features, and improved inter-arrival time series	Random forest, decision tree, and GBDT	○

narrow down the scope before the final traffic classification. Besides, they need to perform traffic segmentation to locate different traffic bursts, where each burst represents a group of adjacent packets that support an application event.

In this subsection, we introduce representative application usage inference approaches, demonstrating their applicable scenarios and methodologies (Table 9 shows a comparison).

2.4.7.1 Messager/Online social network usage inference.

User activities on messaging or OSN applications are very private and sensitive. However, although being encrypted, a third party can still infer the rough messaging/OSN activities that users have performed only through content-agnostic network traffic data.

Back in 2009, Schneider et al. [336] investigated OSN usages from the perspective of network traffic for four different platforms—Facebook, LinkedIn, Hi5, and StudiVZ. The authors studied how users actually interact with OSNs by extracting clickstreams from passively monitored network traffic. They found that different OSN operations (e.g.login, open friend list, logout, select profile, etc.) will trigger statistically different network traffic. This research later lead many researchers to dig deeper into using the traffic differences to classify different user actions on OSNs. Coull et al. [111] analyzed the network traffic of encrypted messaging services such as Apple iMessage. The authors demonstrated that an eavesdropper can learn information about user actions (e.g.control, read, start, stop, image, and text), the language of messages, and even the length of those messages with greater than 96% accuracy simply by observing the sizes of encrypted packets. They used three algorithms to perform the inference—linear regression, Naïve Bayes, and rule lookup table. However, they only evaluated their approach in closed-world environments with a small dataset. Fu et al. [166] extended the inference to more messaging applications (i.e.Wechat and WhatsApp) and more activities (e.g.stream video call, news feed, location sharing, etc.). By segmenting Internet traffic into sessions with a number of dialogs, extracting discriminative features from the perspectives of packet length and time delay, and leveraging multiple machine learning models to conduct the classification, the proposed approach can achieve 96% and 97% accuracy in WeChat and WhatsApp respectively. Liu et al. [255] further extended the inference coverage to more OSN applications (e.g.Facebook, Wechat, and WhatsApp) and evaluated their approach in a real-world environment with real-time traffic data streaming. Real-world evaluation is essential to reveal the true performance and efficacy of application usage approaches, but many approaches were only evaluated through closed-world off-line cases, leaving the inference throughput and abilities to handle noise mysteries. Feng et al. [152, 145] developed and evaluated their OSN usage inference approach in a larger network environment—a campus network. Although their approach is mainly built for social bot detection, it can identify some commonly seen user activities (i.e.posting, reading, liking, etc.) on Twitter and Facebook.

2.4.7.2 Streaming service usage inference. There are a few works focusing on leveraging FGTA to extract behavioral information from network traffic of streaming service (e.g.VoIP, audio streaming, and video streaming). Researchers have demonstrated the feasibility of revealing voice information from encrypted VoIP conversations or identifying encrypted video streams [299].

Wright et al. [417] demonstrated that when the audio is encoded using variable bit rate codecs, the lengths of encrypted VoIP packets can be used to identify the phrases spoken within a call. By leveraging a HMM, the authors indicated that an eavesdropper can identify phrases from a standard speech corpus within encrypted calls with an average accuracy of 50%, and with accuracy greater than 90% for some phrases. Schuster et al. [337] demonstrated that many video streams are uniquely characterized by their burst patterns, and classifiers based on CNN models can accurately identify these patterns given very coarse network measurements. The authors only extracted features from flow attributes, such as inbound/outbound bytes per second, inbound/outbound packet per second, and inbound/outbound average packet length. They have examined this approach on Netflix, YouTube, Amazon, and Vimeo.

2.4.7.3 General-purpose application usage inference. The approaches discussed in this subsection aim at inferring all types of application-layer events rather than only recognizing certain event categories.

Conti et al. [109, 108] analyzed encrypted mobile traffic to infer user actions on Android devices, such as email exchange, posting a photo online, publishing a tweet, etc. They extracted features from TCP/IP packet fields (e.g.IP address, port number, packet size, direction, and timing) and use a random forest to perform the inference. They trained and evaluated their approach using a dataset that is associated with several Android applications with diverse functionalities, such as Gmail, Facebook, Twitter, Tumblr and Dropbox. The evaluation results demonstrate that it can achieve more than 95% of accuracy and precision for most of the actions within the dataset. However, this approach was not evaluated in the real-world environments. In 2016, Saltaformaggio et al. [333] proposed NetScope, a framework that can perform robust inferences of user activities for both Android and IOS devices by only inspecting IP packet headers. NetScope leverages a K-means model and an SVM model to learn and detect network traffic generated by different application behaviors. By testing the approach in a lab environment, the authors demonstrated that despite the widespread use of fully encrypted communication, NetScope can distinguish subtle traffic behavioral differences between user activities (e.g.Instagram browse versus post, Yelp browse versus search, Facebook feed versus post, etc.). Papadogiannaki et al. [297] further pushed application usage inference to a much larger scale. They proposed OTTer, a highly scalable engine that identifies fine-grained user actions (e.g.voice call, video call, messaging, etc.) in popular Over-The-Top mobile applications, such as WhatsApp, Skype, Viber, and Facebook Messenger with encrypted network traffic connections. By evaluating OTTer is a real-world test bed, the authors demonstrated that it can operate at traffic loads with an average of 109 Gbps.

2.4.7.4 Others. There are a few application usage inference approaches tackling different problems. For instance, Yan et al. [428] segmented the network traffic into several bursts and trained a random forest model to identify red packet transactions and fund transfers in Wechat; Wang et al. [409] proposed an approach to identify the mobile payment applications from traffic data, then classify specific actions (e.g.transfer payment, transfer receipt, QR code payment, etc.) on the mobile payment application, and finally, detect the detailed steps (e.g.click the button, receive the fund, open the red packet, etc.) within the action; Jiang et al. [218] studied encrypted remote desktop traffic and found that an eavesdropper can reveal application usage information (e.g.reading documents, surfing webs, editing documents, etc.) due to side-channel privacy leakage. Wang et al. [408] aimed at identifying DApp (e.g.Superrare, Editional, John Orion Young, etc.) user behaviors (e.g.open DApps, open market, view detail, etc.) on Ethereum by using random forest, decision tree, and gradient boosting decision tree (GBDT).

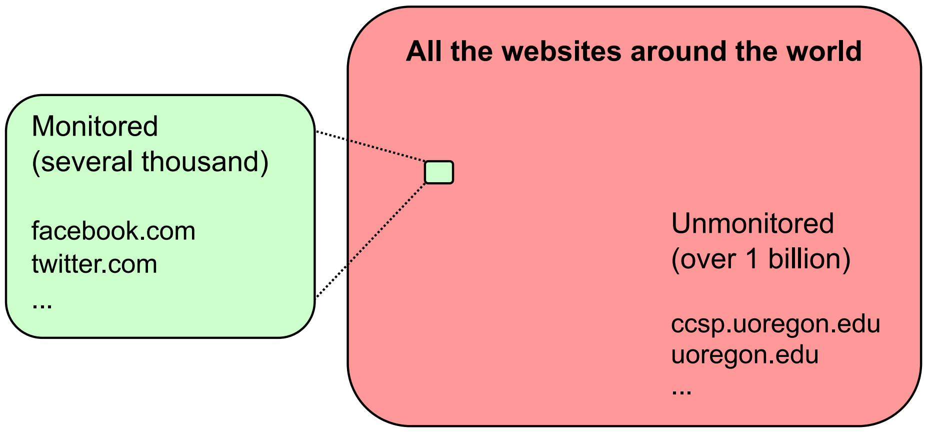


Figure 11. Training data coverage for WFP.

2.5 Limitations

Although it seems effective in inferring different high-level, fine-grained behaviors, FGTA still has many limitations. In fact, FGTA approaches are far from what they have promised in the real world, and their efficacies depend on many conditions. In this section, we discuss the limitations of FGTA.

2.5.1 Coverage of train data. As most FGTA approaches are based on machine learning algorithms or prior knowledge about specific traffic, the efficacy of such approaches is highly dependent on the coverage of training datasets or rules learned beforehand. Unfortunately, existing datasets or rules can only represent a small fraction of real-world scenarios. It is actually impossible to collect a dataset to cover all possible scenarios. Take the WFP attack discussed in Section 2.4.3 as an example, as shown in Figure 11, state-of-the-art datasets from public repositories can only cover less than 0.001% of all websites around the world. FGTA approaches built upon such datasets then have little effect in practice. Furthermore, network traffic of websites, applications, or OSes is dynamic. For instance, the layouts of Facebook websites have been changed for several times since its birth, and so has the network traffic associated with Facebook. Therefore, a FGTA approach that worked before may no longer be effective, if we do not update its classification model with the latest training datasets.

2.5.2 Uncertainties in real-world environments. As can be seen from our previous discussion (Section 2.3.4 and Section 2.4), many approaches were only evaluated in closed-world environments, which means they were only tested with a small amount of labeled traffic, with a little noise or without noise. Such closed-world evaluations cannot objectively reveal the efficacy of proposed approaches in the real world. Network traffic in real-world environments can be very quite different from traffic in laboratory environments:

- Real-world network configurations can be complicated, with traffic going through NATs, Wi-Fi connections, or special middle boxes. All these factors can significantly change the original traffic characteristics.
- Edge users have different habits of using web applications. Some may send traffic with VPN, Shadowsocks, or Tor. Although many FGTA approaches claim to be effective even with traffic tunneling techniques, many researchers found their efficacy will actually be reduced under such circumstances [444].
- The ratio of different network traffic in the real world is different from that in the laboratory environment, making accuracies obtained from closed-world evaluations hardly representative.

Therefore, real-world evaluations or large-scale pilot studies are essential for developing and polishing a usable FGTA approach.

2.5.3 False alarms. FGTA aims to identify specific types of user activities from network traffic. Usually, the analysis object only occupies a very tiny proportion of the whole traffic (e.g.less than 0.01%). Thus, a very small false positive rate can be amplified in deployment, making the proposed FGTA approach hardly usable.

2.5.4 Integrity of network traffic.

As discussed in Section 2.2, the integrity of the network traffic collected in the real-world cannot be guaranteed. The traffic flow can be asymmetric or highly sampled, which will certainly reduce the efficacy of existing FGTA approaches. However, FGTA with incomplete network traffic was not widely discussed in existing papers.

2.6 Countermeasure

Besides the aforementioned limitations of FGTA approaches, Internet users can also adopt various tricks or methods to escape inference. From the perspective of illegitimate users, these countermeasures can make them stay stealthy and avoid being discovered when conducting malicious activities. On the other hand, legitimate users can also leverage these countermeasures to perturb FGTA, thereby protecting their privacy. This section investigates FGTA countermeasures, comparing their efficacy and use cases.

Naïve countermeasures send individual or aggregated traffic through encrypted channels to escape the inferences of traditional TA approaches, such as VPN, Shadowsocks, and Tor. However, these approaches are proven to be vulnerable to many FGTA approaches [127, 65, 133, 99]. Therefore, people began to modify the features of traffic flows to perturb FGTA approaches' classification models. Such perturbations can be conducted from either network layer or application layer [122]. Table 10 shows a comparison of some well-known countermeasure approaches.

2.6.1 Network-layer Countermeasures.

Network-layer FGTA countermeasures directly modifying the network traffic by adding padding packets,

Table 10. Comparisons of selected well-known FGTA countermeasures (*None*: 0; *Low*: 0-30%; *Medium*: 30%-60%; *High*: more than 60%).

Category	**Approach**	**Usage Scenarios**	**Time Overhead**	**Bandwidth Overhead**	**Additional Requirements**
Network-Layer	Pinheiro et al. [308]	All web applications	None	Medium	Middlebox and SDN controller
	FRONT and GLUE [178]	Tor	None	Low	None
	Traffic morphing [418]	All web applications	None	Low	Knowledge about other traffic classes
	Walkie-Talkie [404]	Web browsing	Medium	Low	Knowledge about some web traffic
	WTF-PAD [222]	Web browsing	None	Low	None
	Liberatore et al. [252]	All web applications	None	High	None
	BuFLO [136]	Web browsing	High	High	Network Transfer with fixed rates
	TrafficSliver [122] (network-layer mode)	Tor	None	None	Multiple entry ORs in Tor network
Application-Layer	HTTPOS [263]	Web browsing	None	Low	None
	TrafficSliver [122] (L7 mode)	Tor	None	None	Multiple entry ORs in Tor network
	LLaMA and ALPaCA [95]	Tor	Server-side: Medium; Client-side: Low	Server-side: Medium; Client-side: Low	None

changing packet bytes, or delaying existing packets, thereby obfuscating specific features that FGTA approaches rely on, making the current traffic look like other activities', or regularizing the traffic patterns of different applications [178]. Such approaches usually come with some side effects. They might increase the overheads of the network system, including time overhead, bandwidth overhead, and potentially computational overhead.

Among all the network-layer countermeasures, traffic obfuscation is the most classic approach. Back in 2006, Liberatore et al. [252] leveraged per-packet padding (i.e.increasing the bytes of packets) in an attempt to defeat host profiling system. They found that per-packet padding is reasonably effective, which can lower predictive accuracy to less than 8% with a cost of increasing traffic volume by 145%. However, per-packet padding cannot defend against many WFP attacks [136, 192] because this approach still preserves some key traffic features that can help classify the traffic. To fix the drawbacks, WTF-PAD [222] extends per-packet padding to link-based padding to modify more traffic features. It detects large time gaps between packets and covers them by adding dummy packets. Further, to obscure traffic bursts, it also adds delays between packets to make them statistically different. Due to its low computational overhead and time overhead, WFP-PAD has been used in many real-world FGTA defense systems [174, 26]. Still, WTF-PAD leaks a portion of information in transmission and can be broken by some FGTA approaches [348, 250]. Gong et al. [178] proposed FRONT and GLUE. FRONT focuses on obfuscating the trace front with dummy packets. It also randomizes the number and distribution of dummy packets to impede the attacker's inferring process. GLUE adds dummy packets between separate traces so that they appear to the attacker as a long consecutive trace, making the attacker unable to find the start or end points.

Compared with traffic obfuscation that freely modifies traffic features, traffic confusion mimic other groups of traffic to let FGTA approaches generate wrong outputs, which is sometimes more effective, especially when defending against WFP attacks. Wright et al. proposed traffic morphing [418]. It can thwart statistical TA approaches by morphing one class of traffic to look like another class using convex optimizations. Although it cannot defend against some types of FGTA approaches [136, 192], this approach inspired many subsequent countermeasure approaches. For example, Glove [287] first leverages a clustering algorithm to group web pages with similar traffic, and then inserts only a small amount of dummy traffic to hide the web page traffic in a close group; Supersequence [402] also clusters network traffic traces of different web pages and extracts the shortest common supersequence to cover current web traffic; Walkie-Talkie [404] modifies the browser to communicate in half-duplex mode (buffer traffic and send in bursts) rather than the usual full-duplex mode (immediately send available data). By combining with dummy packets, Walkie-Talkie can modify the traffic of monitored sensitive pages and benign non-sensitive pages, so that each page's packet sequences are exactly the same (each packet has the same timing, length, direction and sequence number). However, a traffic-confusion-based approach requires a priori knowledge about popular web pages' network traffic. It cannot tackle traffic of dynamic content or unpredictable activities. Moreover, such approaches can lead to noticeable computational overhead.

Another obfuscation direction is to regularize the network traffic, making different groups of traffic have relatively uniform patterns. For instance, Buffered

Fixed-Length Obfuscation (BuFLO) [136] obfuscates page transmissions by sending packets of a fixed size at a fixed interval and using dummy packets to both fill in and potentially extend the transmission. Thus, the traffic generated by different websites has a similar continuous traffic flow. However, BuFLO can cause very high time and bandwidth overhead, sometimes can even bring congestion problems to the network [82]. To alleviate the problem, Congestion-Sensitive BuFLO (CS-BuFLO) [80] was proposed to vary the packet transmission rate. Tamaraw [81] achieves a better security/bandwidth trade-off by using smaller fixed packet sizes and treating incoming and outgoing packets differently to avoid unnecessary padding and dummy traffic. DynaFlow [259] morphs packets into fixed bursts, dynamically changes packet inter-arrival times to generate constant traffic flows, and pads the number of bursts. Theoretically, DynaFlow leads to less network overhead compared with BuFLO, CS-BuFLO, and Tamaraw.

The recent development of FGTA countermeasures mainly focuses on two aspects:

1. The countermeasure should lead to nearly zero overhead to both the data plane and the endpoints.

2. The countermeasure should be applicable to various web applications (e.g.web page visiting, video streaming, VoIP, etc.) and scenarios.

For instance, Henri et al. [196] split traffic exchanged between the user and Tor nodes over two different, unrelated network connections (e.g., DSL, Wi-Fi, or cellular networks) to protect against FGTA by a malicious ISP; TrafficSliver [122] limits the data a single observation point can observe and distorts repeatable traffic patterns exploited by FGTAs with user-controlled splitting of traffic over multiple

Tor entry nodes. TrafficSliver also offers an application-layer solution, which will be discussed in Section 2.6.2; Wang [401] points out that an attacker may only need to successfully identify a single web page (which they define as the one-page setting) in reality, and a WFP countermeasure must still thwart that attempt. Based on this assumption, Wang fortifies WFP countermeasures by exploring randomness and regularization options for several existing countermeasures. To protect IoT networks, Pinheiro et al. [308] implement a middlebox to modify the outbound and inbound traffic's packet size. They also leverage an SDN application to obtain information of network traffic from both sides (source and destination) to manage the size-based padding mechanism.

2.6.2 Application-layer Countermeasures. Unlike network-layer countermeasures that directly modify network traffic to cover user activities, application-layer countermeasures use dummy applications to generate unnecessary traffic, thereby indirectly perturbing FGTA approaches. However, most application-layer countermeasures are limited in covering traffic of web page being visited.

Panchenko et al. [296] proposed a browser plug-in that adds traffic noise by loading another random web page in parallel. However, it may fail to defend against some WFP attacks if users lower the page loading frequency to decrease the bandwidth overhead [402]. Another Tor-based countermeasures approach [271] randomizes the order of requests for embedded website content and the pipeline size (i.e.the number of requests processed in parallel) to perturb WFPs. Cherubin et al. [95] propose LLaMA and ALPaCA, defenses for client side and server side. LLaMA reorders outgoing HTTP requests by randomly delaying them and adding dummy HTTP requests. On the server side, ALPaCA conducts traffic morphing by padding web objects of a page and inserting invisible dummy web objects. The three methods above only work in Tor environments.

HTTP Obfuscation (HTTPOS) [263] is countermeasure that can be used in environments other than Tor. By modifying HTTP requests and basic TCP features, it manipulates four fundamental network flow features, including packet size, web object size, flow size, and timing of packets. It can also modify and reorder HTTP headers and insert dummy HTTP requests. Another general countermeasure is TrafficSliver's application-layer defense [122]. This approach is on the client side. By sending single HTTP requests for different web objects over distinct Tor entry nodes, this application-layer defense can reduce the detection rate of WFP classifiers by almost 50 percentage points.

2.7 Future Research Direction

Although FGTA has been developed for decades, there still exists room for further development, enhancement, and exploration. In this section, we discuss avenues for future research according to our observations about recent research trends, existing literature, industry deployments, and major problems to be solved in this domain.

2.7.1 Improvement of Analysis Efficacy and Coverage.

FGTA has been used in many different subfields of computer network, including attack detection, traffic measurement, side-channel attack, and network management, etc. Researchers have constructed myriad analysis models and collected plenty of datasets specifically for different categories of tasks. But there are still many use cases or scenarios that have not been covered by existing approaches. For example, with the raising of Unmanned Aerial Vehicle (UAV), FGTA can be potentially refined for UAV anomaly detection [164]; FGTA can also be adopted to monitor and safeguard network traffic of automatic vehicles; with the emergence of new forms of applications, attacks, and communication protocols, existing FGTA approaches may not be able to handle today's traffic. Therefore, researchers can gather more updated traffic datasets to enhance the coverage of existing FGTA approaches, so that they can be used in more types of tasks and scenarios.

In addition, the efficacy of many current FGTA approaches are not ideal for real-world deployments. Depending on the observation points, FGTA approaches may easily see millions of traffic flows over a short time period in the real world. Under such circumstances, an FGTA approach could generate large numbers of false positives or false negatives, even if it achieves more than 95% accuracies in closed-world evaluations. Thus, increasing the efficacy of FGTA is a timeless topic for researchers and developers.

2.7.2 Evaluation Enhancement. As we elaborate in Section 2.5, current closed-world evaluation methods are far away from revealing an FGTA approach's real capability and many open-world evaluations are not very standardized and effective. It is therefore suitable to propose a new, operable, and effective evaluation paradigm for FGTA. Such an evaluation paradigm should contain a testing dataset similar to a real-world test case in terms of volume, environment, and data distribution. Simultaneously, the dataset should have comprehensive labels for almost all traffic flows, not only for analysis targets. This can be achieved by either constructing a large scale sandbox to simulate and collect all types of traffic from a white box view, or collect a large-scale, real-world traffic dataset and carefully label it using knowledge of endpoints from all perspectives. Besides, the testing data portion that is visible to the observation point should be consistent with the real-world deployment conditions.

2.7.3 Dealing with Complex Network Environments.

In real-world deployments, the network environments and configurations can be different from researchers' assumptions. The following factors were not widely discussed in previous papers, but can be common for network service providers.

- Many observation points can only see asymmetric network traffic, which can challenge most FGTA approaches.
- Some networks are composed of multiple subnets, including but not limited to wireless network, optical network, or radio frequency network. Traffic flows collected from such a network can have different delays and congestion control mechanisms. Tackling this type of traffic can be challenging.
- Due to deployments of modern traffic engineering approaches, traffic captured from some observation points is dynamic, posing difficulties to many FGTA methods.

We believe designing and implementing new FGTA approaches that can work under these circumstances are directions worthy of future research.

2.7.4 Integrating FGTA into Other Analytical Systems.

Information contained in network traffic is essentially limited. Even though FGTA can already reveal considerable amount of information, the detailed behavior models of endpoints are still hidden behind the curtain. To more comprehensively investigate the network situation, researchers can try to combine FGTA with information from other dimensions (e.g.application-layer activities, server specifics, hardware conditions, etc.), which can provide a better situational awareness. So far, there are a few researches that combine TA with information from other layers for more accurate attack/anomaly detection and timely threat response (e.g.[154, 406, 69]). Researchers can push this idea forward by further integrating FGTA into this idea.

Furthermore, cyber threat intelligence (CTI) [57], allowing entities to share attack/anomaly information with trusted partners and peers, is becoming a powerful tool to quickly and accurately tackle intractable attacks. By embedding results from FGTA into CTI systems, participating entities can raise awareness of the current situation, thereby more quickly responding to incoming attacks. Designing attack defense systems with both FGTA and CTI is thus a promising research direction.

2.8 Related Work

In this section, we introduce related work on TA and compare our paper with them. Table 11 summarizes the most related and representative ones.

Our work differs from existing works in the following aspects:

– we have a clear and focused survey topic: the whole paper focuses on FGTA, which aims to analyze network traffic to deduce information related to high-layer activities, fine-grained user behaviors, or application-layer message content;

– to investigate FGTA comprehensively, our paper utilizes multiple methodologies, including literature survey, summarization, and taxonomization, to cover different subjects, such as traffic capture, application identification, website fingerprinting, countermeasures, etc. Most existing related survey papers only cover a subset of the aforementioned topics;

– although studied for many years, TA is still iterating rapidly and continuously, especially for FGTA. Compared with other earlier literature,

Table 11. Overview of related literature (○: not include; ◐: partially include; ●: include).

Ref.	Year	Summary	Focus	Subjects covered			
				General TA	FGTA	Traffic capture	Counter-measure
This paper	2022	A survey of FGTA, which aims to analyze network traffic to deduce information related to high-layer activities, fine-grained user behaviors, or application-layer message content.	FGTA	◐	●	●	●
[342]	2022	The most recent survey on achievements in machine learning-powered encrypted traffic analysis, including the workflow, feature extraction, and algorithms.	Machine-learning-based encrypted TA	◐	◐	◐	◐
[219]	2022	A survey consists of an analysis of IoT traffic data acquisition approaches, a classification of public datasets, a literature evaluation of IoT traffic processing, and a comparison of ML approaches for IoT device classification.	Machine-learning-based IoT TA	○	◐	●	○
[299]	2021	A survey of literature that deals with network traffic analysis and inspection after the ascent of encryption in communication channels.	Encrypted TA	◐	◐	○	●
[419]	2021	An extensive analysis of the communications channels of 32 IoT consumer devices, including traffic measurement and modelling.	IoT TA	◐	◐	◐	○
[363]	2020	This survey looks at the emerging trends of network traffic classification in IoT and the utilization of traffic classification in its applications. It also compares the legacy of traffic classification methods.	IoT TA	●	◐	◐	○
[137]	2019	This survey mainly focuses on approaches and technologies to manage the big traffic data, additionally briefly discussing big data analytics (e.g., machine learning) for the sake of TA.	General TA	●	◐	●	○
[107]	2018	A review of works that contributed to the network traffic analysis targeting mobile devices, including a systematic classification of such works according to their goal, traffic capture point, and targeted platforms.	Mobile device TA	◐	◐	◐	●
[294]	2018	A systematic review based on the steps to achieve traffic classification by using machine learning techniques, including their workflow, feature extraction, deployment, etc.	Machine-learning-based TA	◐	◐	○	○
[284]	2016	An examination of the literature on analyses of mobile traffic collected by operators within their network infrastructure.	Mobile device TA	◐	◐	●	○
[382]	2015	A survey of approaches for classification and analysis of encrypted traffic, including widespread encryption protocols and payload and feature-based classification approaches.	Encrypted TA	◐	◐	○	○
[162]	2014	A survey in which a complete and thorough analysis of the most important opensource deep packet inspection modules is performed.	Payload-based TA	◐	○	◐	○
[177]	2013	A survey of peer-to-peer traffic detection and classification, with an extended review of the related literature.	P2P TA	◐	○	○	○
[83]	2009	A survey explains the main techniques and problems known in the field of IP traffic analysis and focuses on application detection.	General TA	●	○	●	○
[90]	2009	A report attempts to provide an overview of some of the widely used network traffic models, highlighting the core features of the model and traffic characteristics they capture best.	General TA	◐	○	○	○

this paper sorts out and examine the most recent development of FGTA at the time of writing this paper.

In the early development of TA (i.e., before 2010), the survey papers in this field mainly focus on traditional TA [90, 83, 141], including protocol-level traffic classification, TA approaches based on deep packet inspection (DPI), distinguishing server or peer-to-peer nodes from clients, and coarse grained application identifications.

Later, due to increasingly diverse web-applications and widespread use of traffic encryption, there is an increasing need for more sophisticated TA approaches to monitor and analyze the modern networks. Meanwhile, the evolution of classification algorithms and easy access to big data also effectively stimulate the development of TA. Therefore, survey papers began to examine works that leverage big data [137] or machine learning [294] to tackle TA.

On the other hand, the network is also becoming more and more specialized, which has spawned many TA approaches with specific design goals. To track such a trend, many of the recent survey papers only survey a certain type of TA approaches, such as TA for IoT devices [363, 419], encrypted TA [342, 299], TA for mobile devices [284, 107], etc. Similar to these papers, our work focuses on a new and specific topic—FGTA, which has not been systematically studied before.

2.9 Conclusion

With the increasing complexity of network transmission technology, FGTA is becoming a crucial tool to gain a finer granularity of visibility over the network. From the perspective of attackers, it can analyze the content-agnostic metadata and statistical information of network traffic to infer the website visited by users, estimate locations of traffic sender, or decode the video content streamed in the link. As for the network administrators, FGTA can be used to detect application-layer threats even with layer 3 or layer 4 data, investigate quality of experience without collect sensitive user data, or perform fine-grained traffic measurement to better configure the network.

In this paper, we analyze literature that deal with FGTA to help researchers and developers learn the latest developments in this area. After comparing different FGTA approaches by their use cases, we found that most existing approaches are based on machine learning or high-dimensional clustering. They are effective in capturing the subtle differences between network traffic generated by different activities. However, many FGTA approaches still come with limitations related to training data coverage, false positive rates, and real-world usability. In addition, edge users of the network can adopt a variety of countermeasures to defend against FGTA, with some overheads regarding network bandwidth and delay. Researchers can further study this domain to increase the coverage of FGTA or make the approaches more practical in complex real-world network environments.

CHAPTER III
MEASUREMENT AND CONTENT-AGNOSTIC DETECTION OF CRYPTOJACKING TRAFFIC

In this chapter, we broaden the use case of FGTA to detect malicious activities that were previously challenging to identify by traditional TA-based approaches. Specifically, we employ FGTA to discern cryptojacking traffic within the network, demonstrating its versatility and effectiveness in the realm of cybersecurity. Cryptojacking refers to the act by which computing resources are stolen to mine cryptocurrencies, such as Bitcoin, Monero, and Ethereum. With sophisticated system designs and implementations, we demonstrate that FGTA techniques can distinguish cryptojacking traffic from user-initiated cryptocurrency mining traffic, making it possible to only filter cryptojacking traffic, rather than blindly filtering all cryptocurrency mining traffic as commonly practiced. This fine-grained traffic discrimination is difficult to accomplish through traditional TA methodologies and is also the main contribution of this chapter.

This chapter is derived in part from the following published articles:

– Published as Yebo Feng, Jun Li, and Devkishen Sisodia. "CJ-Sniffer: Measurement and Content-Agnostic Detection of Cryptojacking Traffic." *In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), pp. 482-494*, 2022 [155].

– Published as Yebo Feng, Devkishen Sisodia, and Jun Li. "Poster: Content-agnostic identification of cryptojacking in network traffic." *In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIACCS), pp. 907-909*, 2020 [157].

I am the leading author of the above articles. Most content of this chapter was written by me, and I was responsible for conducting all the system designs, implementations, evaluations, and the presented analyses.

3.1 Introduction

With the frenzy of the cryptocurrency market, cryptocurrency mining (cryptomining) has become a method of making huge profits. In fact, cryptomining is critical in many blockchain-based systems, as it not only provides a means to verify cryptocurrency transactions, but more importantly, also helps establish consensus through different mechanisms [251], such as Proof of Work (PoW) and Proof of Space (PoS). Therefore, to encourage cryptomining, cryptocurrency systems usually reward miners with transaction fees and extra coins. Unfortunately, the lucrative potential of cryptomining has caught the attention of hackers, who compromise personal computers, servers, or even Internet-of-Things (IoT) devices, such as smart TVs, to mine cryptocurrencies (e.g., BTC, XMR) [447]. Such activity is called **cryptojacking**, which is the unauthorized use of someone else's computing resources to mine cryptocurrency. Such a hacker is also called a **cryptojacker** and such a resource is said to be **cryptojacked**.

There are many methods of conducting cryptojacking [204]. For example, a cryptojacker can trick a victim into clicking on a malicious link in an email to download cryptomining scripts onto their computer, infect a website with JavaScript code to automatically run the code by a victim's browser when it visits the website, or compromise a server to stealthily execute cryptomining programs in the background. Although Coinhive, an in-browser mining service provider, was shut down in March 2019, cryptojacking has still been active and evolving [381]. According to the Unit 42 Cloud Threat Report [377], from December 2020 to

February 2021, 17% of organizations with a cloud infrastructure showed signs of cryptojacking, causing significant computing resource abuse and tremendous economic loss. Recently, the SophosLabs team also disclosed that cryptojackers compromised unpatched Exchange servers to mine XMR [76].

There have been many endpoint-based approaches to cryptojacking defense [116, 405, 204]. By monitoring software operations, website visits, or hardware conditions, an endpoint-based approach can often achieve decent accuracy and is usually easy to deploy if resources permit, but it is almost infeasible to deploy it on all endpoints due to user inertia and many endpoints are indeed resource-constrained. To address this issue, researchers and developers proposed network-based approaches to detect general cryptomining activities by analyzing cryptominers' network traffic [210, 87, 301]. These approaches are often deployed only at a network gateway, avoiding the need for every device in the network to deploy a defense solution. However, as application-layer behavior inference through layer-3 network traffic is extremely difficult, whether these network-based approaches are based on IP blocklists or deep packet inspection, or conduct state-of-the-art traffic analysis, they can hardly distinguish cryptojacking from **user-initiated cryptomining**, which is cryptomining performed by legitimate users of computing devices in use. Thus, network-based approaches treat all cryptocurrency mining traffic as either legitimate and allowing all of the traffic, or malicious and dropping all of the traffic.

To fill the missing gap, in this paper, we first thoroughly measure and model the characteristics of cryptomining network traffic, including exploring the tiny differences between cryptojacking traffic and user-initiated cryptomining traffic. We then propose CryptoJacking-Sniffer (CJ-Sniffer), a content-agnostic, easily deployable approach that not only can detect cryptomining activities, but also can distinguish cryptojacking from user-initiated cryptomining, only by analyzing layer-3 network traffic.

CJ-Sniffer operates in three phases. In the first phase, CJ-Sniffer inspects packet sizes of every connection to discard obviously irrelevant connections, which could significantly reduce the data volume for further analysis. In the second phase, CJ-Sniffer detects cryptomining connections by comparing every connection's packet interval distribution with labeled traffic. In the third phase, CJ-Sniffer leverages the long short-term memory (LSTM) model [200] to distinguish cryptojacking connections from user-initiated cryptomining connections. As a result, CJ-Sniffer can efficiently classify network connections into three categories: cryptojacking connections, user-initiated cryptomining connections, and other connections.

Our work makes the following contributions:

– CJ-Sniffer as a content-agnostic, network-based approach to detecting cryptojacking activities is not only efficient since it does not inspect packet payload, but also preserves user privacy by only leveraging a limited amount of anonymized metadata of packets.

– CJ-Sniffer is the first work to distinguish cryptojacking traffic from user-initiated cryptomining traffic. This is important because it is often necessary to only filter cryptojacking traffic while preserving user-initiated cryptomining traffic from a device. While the difference between cryptomining traffic and other network traffic is already challenging to detect at a network gateway, the difference between cryptojacking traffic and user-initiated cryptomining traffic is significantly harder to detect.

– CJ-Sniffer is easy and efficient to deploy. While cryptojacking is rampant, any institution, enterprise, or an individual user can deploy CJ-Sniffer at their network's gateway to safeguard all the computing devices in their network from cryptojacking at line speed.

– We collect, measure, and release the first labeled, packet-level cryptomining traffic dataset to the research community. It contains more than 500 hours of both cryptojacking and user-initiated cryptomining traffic from various types of computing devices.

We evaluated CJ-Sniffer with real-world network traffic and found that its efficacy and efficiency are both high. Even running on an ordinary personal computer, CJ-Sniffer is capable of providing real-time traffic monitoring for an enterprise or campus-level network. To reach a detection accuracy of 95%, it only needs to collect approximately 160 network packets from a cryptojacked device, and approximately 200 network packets to reach an accuracy of 97%, all with nearly zero false alarms.

The rest of this paper is organized as follows. After we outline related work in Section 3.2, we describe our discoveries of cryptojacking and cryptomining traffic properties (Section 3.3). We then illustrate CJ-Sniffer's design in Section 3.4 and evaluate it in Section 3.5. We discuss its limitations, open issues, and potential generalizations in Section 3.6 and conclude the paper in Section 3.7. Of a particular note here is that the research described in this chapter is extended from our early work published in [157] and [155].

3.2 Related Work

In the past few years, researchers have proposed various cryptomining or cryptojacking defense approaches. According to the deployment position of these approaches, they are generally either endpoint-based or network-based, or a combination of the two with multiple deployment positions.

3.2.1 Endpoint-based Cryptojacking Defense. Endpoint-based cryptojacking defense is dominated by various endpoint-based cryptojacking detection approaches. Usually, these approaches can be easily embedded in antivirus tools, system firewalls, or even browsers.

Endpoint-based cryptojacking detection approaches can use different types of input to conduct detection. The most common input is source code of software or websites. For example, CMTracker [204] first uses hash-based and stack-based profiling methods to extract features from websites and then matches the features against hand-crafted rules to identify websites with cryptojacking code; SEISMIC [405] derives semantic signatures from known cryptojacking scripts and then matches running scripts at an endpoint against signatures to detect cryptojacking scripts. Besides, hardware conditions can also be used as input. Gomes et al. [175] extracted features from the CPU usage and used machine learning to detect cryptojacking processes. Tahir et al. [364] proposed a machine learning detection solution based on features from Hardware Performance Counters (HPCs) values. However, hardware-based approaches may generate an excess of false alarms, as many legitimate processes sometimes utilize hardware in a way similar to cryptojacking. To address this issue, CoinPolice [305] actively changes the execution speed of processes, then collects execution traces at various execution speeds, and inputs the collected data into a deep neural network to pinpoint cryptojacking scripts.

These approaches can often achieve a decent accuracy because they have access to a variety of system and software metrics from end-users. However, the

Table 12. Comparisons of selected network-based cryptomining detection approaches (○: not support; ◐: partially support; ●: fully support).

Approach	**Content Agnostic**	**Cryptomining Detection**	**User-initiated Cryptomining v.s. Cryptojacking**
DPI-based solution	○	●	●
Cisco solution [45]	◐	●	○
Munoz et al. [210]	●	●	○
Pastor et al. [301]	●	●	○
Hu et al. [208]	●	●	○
MineHunter [443]	●	●	○
CJ-Sniffer	●	●	●

popularity of these approaches is limited by user habits, as not all users are willing to install cryptojacking detection software and have it monitor their computer thoroughly. Also, some IoT devices, such as smart furniture and health-monitoring devices, have too little resources to deploy these approaches. Additionally, cryptojacking malware is becoming more sophisticated; for example, hackers could obfuscate their code to thwart an endpoint-based approach. It is thus necessary to update an endpoint-based approach frequently.

3.2.2 Network-based Cryptomining Defense. Network-based cryptomining defense operates at certain vantage points of a network so that any devices within the network can be protected. During the early rise of cryptomining, some companies, schools, and institutions deployed some simple network-based approaches to guard their computing devices. For example, network administrators can block the IP addresses of confirmed mining pools and websites hosting cryptomining code [437, 224, 45]; or, intrusion detection systems can conduct deep packet inspection (DPI) to discover specific cryptomining text strings in packet payloads [124]. These approaches are easy to develop and deploy, but they can only offer preliminary defense against cryptomining due to their low levels of accuracy.

Later, as network traffic analysis techniques [299] evolved, several projects began to detect cryptomining activities with content-agnostic traffic flows. Munoz et al. [210] inspected network flows in NetFlow or IPFIX format with some learning-based algorithms (e.g., SVM, CART, C4.5, and Naïve Bayes) to detect cryptomining activities. Caprolu et al. [88] built a random-forest-based framework to classify network traffic related to pool mining, solo mining, and activities from active full nodes. Pastor et al. [301] extracted several features from NetFlow data and leveraged deep learning models to detect encrypted cryptomining malware connections; Hu et al. [208] indicate that using random forest with extracted discriminative network traffic features can accurately and efficiently detect cryptomining traffic; MineHunter [443] detects cryptomining connections by analyzing packet intervals with a similarity calculation algorithm based on credible probability estimation. These network-based approaches come with many advantages: (1) once deployed in a network, they can protect all users inside the network; (2) they do not analyze message content from users, thus protecting user privacy; (3) they have a higher throughput than DPI methods due to their smaller input size (network flow records vs. packet trace with payloads).

However, none of the aforementioned approaches distinguish between *cryptojacking* and *user-initiated cryptomining* activities (as demonstrated in Table 12). They cannot meet the desire of many networks to forbid only cryptojacking traffic but still allow the user-initiated cryptomining traffic.

3.2.3 Hybrid Cryptojacking Defense.

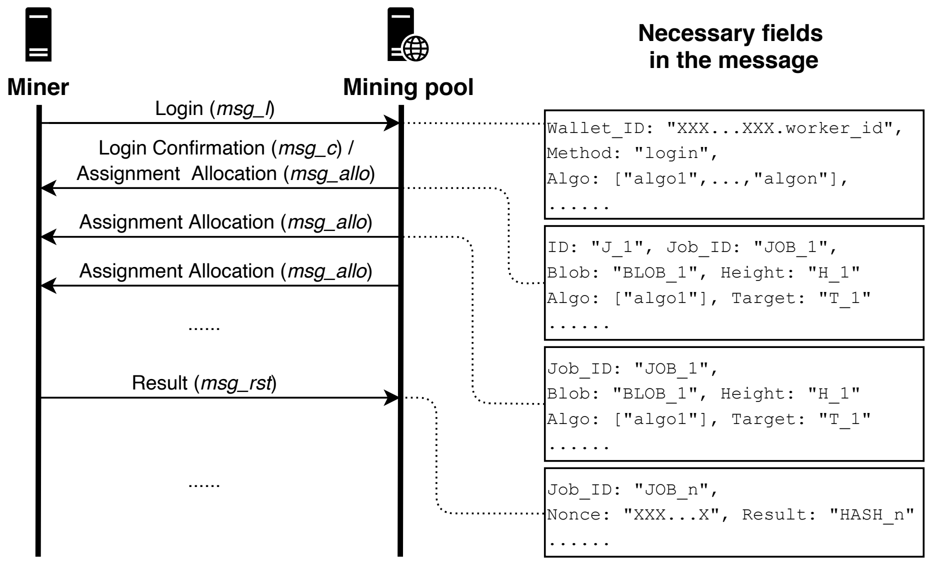
Hybrid cryptojacking defense combines both endpoint-based and network-based approaches. For example, Gomes et al. introduced CryingJackpot [176], which first extracts features from both network traffic flows and endpoint operating system logs and then utilizes unsupervised machine learning algorithms to discover cryptojacked devices with a high F1-Score. Although hybrid cryptojacking defense can achieve robust and accurate detection results on different types of cryptojacking activities, it requires a variety of data to operate, making it challenging to deploy in the real world.

3.3 Measurement of Cryptomining and Cryptojacking Traffic

As discussed in Section 3.1, we classify cryptomining activities into two categories: (1) *user-initiated cryptomining*, refers to the cryptomining conducted by the owner or legitimate user of the computing device; (2) *Cryptojacking*, refers to the cryptomining conducted by attackers using stolen computing resources. In this section, we study and measure the network traffic generated from both types of cryptomining activities. The design and evaluation of CJ-Sniffer is based on these measurement results.

We mainly investigate the mining traffic related to XMR [234], which dominates the cryptojacking [71] campaigns in today's Internet. XMR applies CryptoNight [235] as its hash function, which is ASIC resistance and enables cryptojackers to obtain reasonable profits through different types of computing devices [212]. Conversely, other cryptocurrencies are much harder to mine with ordinary computing devices, making them less profitable for cryptojackers. In addition, other cryptocurrencies that employ PoW consensus mechanisms, will generate similar cryptomining traffic to that of XMR. We further demonstrate this point in Section 3.5.3 and show that by studying the cryptomining traffic of XMR, we can detect many other PoW-based cryptomining traffic

3.3.1 In-depth analysis of general cryptomining traffic. In PoW-based cryptocurrency systems, cryptominers need to participate in puzzle solving competitions to obtain rewards or transaction fees. Due to the exceptional difficulty of solving these puzzles, individual miners with limited computing capacities, particularly the cryptojacked devices, have to join mining pools to ensure stable and prompt profits [256]. Thus, the traffic between miners and the mining pool is the key to detecting cryptomining.


Miner
Mining pool
Necessary fields in the message
Login (*msg_l*)
Login Confirmation (*msg_c*) / Assignment Allocation (*msg_allo*)
Assignment Allocation (*msg_allo*)
Assignment Allocation (*msg_allo*)
......
Result (*msg_rst*)
......

```
Wallet_ID: "XXX...XXX.worker_id",
Method: "login",
Algo: ["algo1",...,"algon"],
......
```

```
ID: "J_1", Job_ID: "JOB_1",
Blob: "BLOB_1", Height: "H_1"
Algo: ["algo1"], Target: "T_1"
......
```

```
Job_ID: "JOB_1",
Blob: "BLOB_1", Height: "H_1"
Algo: ["algo1"], Target: "T_1"
......
```

```
Job_ID: "JOB_n",
Nonce: "XXX...X", Result: "HASH_n"
......
```

Figure 12. Network messages generated by cryptomining and their necessary fields.

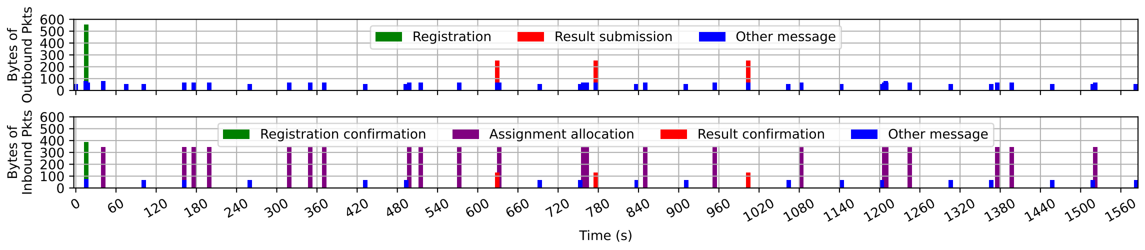


Figure 13. Visualized network packets between an XMR mining pool and an XMR miner. Other messages refer to packets that do not carry meaningful payloads, such as TCP SYN and FIN packets.

For coordination, the mining pool and miners need to obey certain protocols, such as Stratum [319], to register nodes, distribute tasks, and submit results. Figure 12 illustrates the mining process and messages between miners and the mining pool. No matter what types of protocols the mining pool utilizes, there should be at least four types of messages to cover the necessary mining operations:

– The login or registration message msg_l, enabling miners to join the mining pool, can have 75 to 600 bytes per message.

– The login confirmation message msg_c, confirming the login status, sometimes comes with an assignment allocation message.

– The assignment allocation message msg_{allo}, allocating the most recent mining task to the miner, should at least have 285 bytes.

– The result message msg_r, returning calculated results to the mining pool, usually has more than 200 bytes.

Besides, the login message and confirmation message often appear only once during each connection. Thus, assignment allocation messages dominate the inbound traffic and result messages dominate the outbound traffic during the whole cryptomining process. Figure 13 illustrates communication between a miner and mining pool. Excluding TCP control packets such as SYN and ACK packets, we can see that the inbound traffic is mainly composed of assignment allocation messages and the outbound traffic is mainly composed of result messages.

The size of the packet in the cryptomining traffic is distinctive. Figure 14 illustrates packet size ranges of cryptomining traffic and some contrast traffic (collected from our lab and campus network). We can see that the sizes of cryptomining packets are more monotonous compared with others, since the same type of cryptomining messages usually have similar lengths. In addition, cryptomining packets generally have smaller sizes compared with other types of traffic. For most web applications, their maximum packet sizes are usually subject to the network's maximum transmission unit (MTU), which is usually around 1500 bytes. However, the information in each cryptomining operation cannot fill even half of the MTU. Besides, due to timeliness requirements, the miner or mining pool cannot bank messages and send them in a single packet. Thus, all the cryptomining packets are relatively small in size.

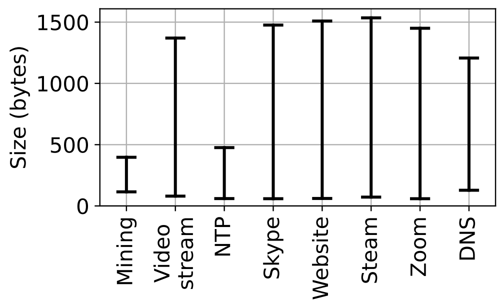


Figure 14. Range of packet sizes of different types of traffic.

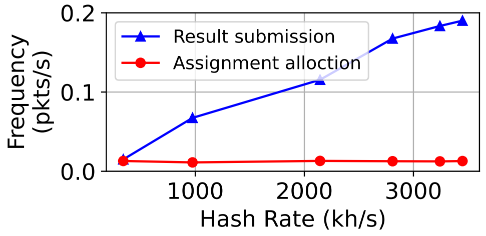


Figure 15. Frequency of different types of cryptomining packets under different hash rates (captured using i7 8700k CPU and XMRig [20]).

Moreover, the frequency of inbound packets is more stable than that of outbound packets. This is because mining tasks expire quickly with the growth of the blockchain. The mining pool needs to keep sending assignment allocation messages (inbound packets) with the growth of the blockchain to ensure that miners always have valid up-to-date tasks. In the short term, the speed of blockchain expansion is very stable, therefore the frequency at which assignment allocation messages (inbound packets) are generated is also stable. On the other hand, the frequency of result submission messages (outbound packets) is related to the hash rate—the speed at which a device is completing an operation in the cryptomining code. The higher the computing performance of a device, the higher its hash rate, and the higher the frequency it sends result messages (outbound packets) to the mining pool. Figure 15 illustrates this pattern, where the frequency of outbound packets is proportional to the hash rate while the frequency of inbound packets is relatively stable. It's also important to note that the frequency of assignment allocation messages doesn't need to be higher than that of result submission messages. Because one assignment can usually derive more than one sub-results to fully complete.

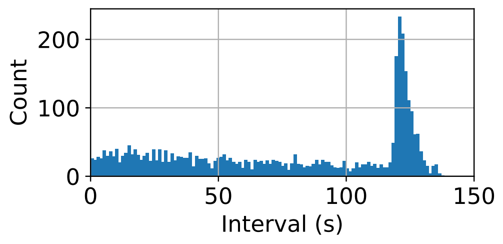


Figure 16. Interval distribution of assignment allocation messages (inbound packets larger than 285 bytes).

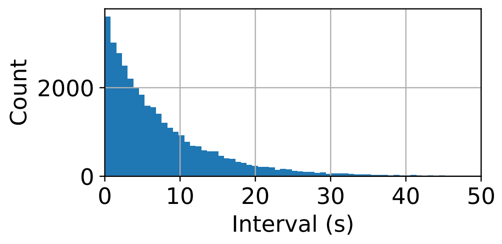


Figure 17. Interval distribution of result submission messages (outbound packets larger than 200 bytes).

Last but not least, the generation of cryptomining packets is not subject to human behavior. The time to generate the next result message depends on the time the device completes the hash backtracking. Analogously, the time to generate the next assignment allocation message depends on the time the mining pool proposes a new task. Hence, the intervals of cryptomining packets exhibit stable and unique distributions. These interval distributions can be treated as fingerprints of cryptomining traffic (illustrated in Figure 16 and 17). We can identify a cryptomining connection by checking whether these interval distributions are obeyed.

3.3.2 Cryptojacking *vs.* user-initiated cryptomining traffic.

After studying cryptojacking activities, we found that they differ from user-initiated cryptomining in the *robustness* of hash rate. This further results in a difference in the network traffic generated by the two.

```
<script src="https://www.XXXpool.com/lib/base.js"></script>
<script>
    var miner=WMP.User('<your-site-key>','<username>',{
        threads: 4,  // number of maximum threads
        autoThreads: true, // adjust the number of threads
           automatically
        throttle: 0.85, // maximum system load
        forceASMJS: false
    });
    miner.start();
</script>
```

Listing 3.1 JavaScript code piece of web-based cryptojacking.

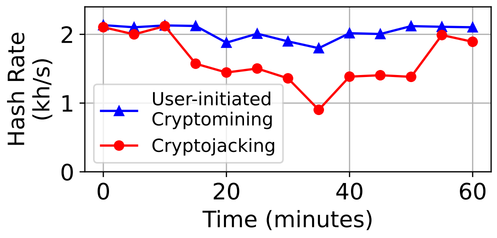
Unlike user-initiated cryptomining, cryptojacking activities have the following features that can lead to unstable mining hash rate: (1) First of all, as injected programs, the execution priority of cryptojacking scripts is usually low, making them easy to be disturbed by the resource manager of operating systems. Modern operating systems will not blindly allocate a huge chunk of resources to random processes (e.g.cryptojacking) because they need to ensure there is enough redundant load to handle high-priority tasks that may occur. Conversely, user-initiated cryptomining usually runs with a high execution priority by system administrators. Even with many background processes, user-initiated cryptomining still has access to a large amount of hardware and software resources. (2) Cryptojackers use stolen computing resources to mine cryptocurrencies. To prevent being discovered by legitimate users of the computing device, cyptojackers usually conduct cryptojacking in surreptitious ways. For example, cryptojackers may dynamically adjust the hash rate of cryptojacked devices to prevent interference with the normal use of users. Listing 3.1 shows part of the JavaScript code that is used for browser-based cryptojacking. We can see that the hacker lets the device automatically adjust the number of mining threads to make sure the system load is under 85%. By doing so, legitimate users will not sense any abnormality in the computing device, thereby allowing the cryptojacking program to run in the background for a long time. Otherwise, the user will quickly sense the abnormality and scan the device for a virus. (3) Moreover, executions of cryptojacking scripts usually rely on executions of existing software running in the system such as the browser, terminal, or Apache server. Due to the uncertainty of human behavior, the execution situation of such software is inconstant, making the computing resources devoted to cryptomining erratic.

We measured the hash rate trends of user-initiated cryptomining and cryptojacking in the real world on the same machines (demonstrated in Figure 18). The measurement results are consistent with our previous analysis that cryptojacking produces unstable hash rates in most cases. Furthermore, according to measurements in Section 3.3.1, this hash-rate instability will generate result submission messages in unstable frequencies.

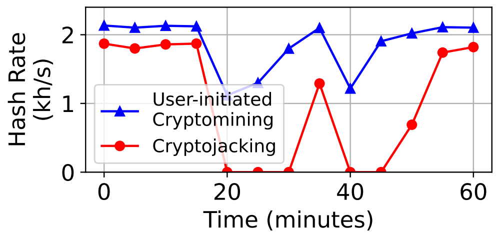
3.4 Design of CJ-Sniffer

In this section, we describe the design details of CJ-Sniffer, an in-network-based cryptojacking traffic detection approach.

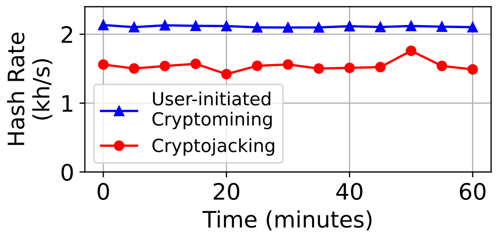
CJ-Sniffer can be deployed within the intrusion detection system (IDS) of any router or switch between the mining pools and devices to be protected. Particularly, CJ-Sniffer functions effectively at the gateway of a network, since this vantage point enables CJ-Sniffer to access complete inbound and outbound



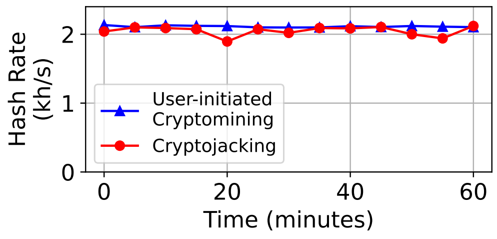
(a) For most cryptojacking activities, the hash rate exhibits irregular changes due to automatic adjustments of computing resources.



(b) Due to minimum memory requirements or time-based resource throttling, hash rates of some cryptojacking activities go to zero from time to time.



(c) Some cryptojacking activities have stable hash rates. However, these hash rates are still lower than that of user-initiated cryptomining using the same hardware.



(d) In rare cases, the attacker utilizes all computing resources of the cryptojacked device, generating hash rates that are similar to user-initiated cryptomining.

Figure 18. Trends in hash rate of different types of cryptojacking activities (extracted from our collected dataset described in Table 13).

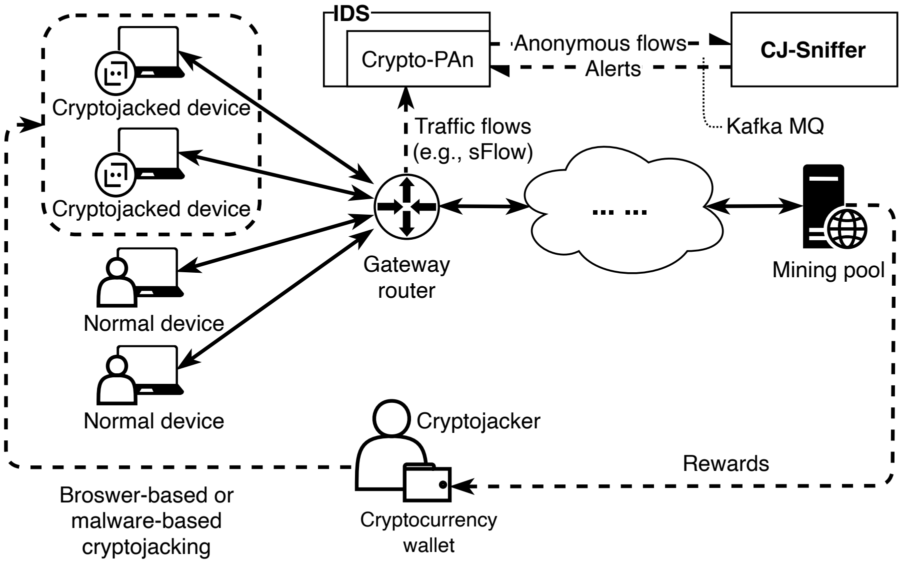


Figure 19. One operational model of CJ-Sniffer, where it is deployed apart from the IDS.

traffic from all the devices in the network. In addition, people can treat CJ-Sniffer as a cloud service and stream the network traffic to it for detection. To prevent leakage of user information in this case, the IP addresses in the traffic flows are anonymized with Crypto-PAn [425] and sent encrypted through a Kafka message queue. Thus, the CJ-Sniffer service provider or any third parties cannot fetch necessary information to trace back to individuals in the network. Figure 19 illustrates the operational model of CJ-Sniffer in this type of deployment,

As stated in Section 3.1, CJ-Sniffer detects cryptojacking activities in three phases. The first detection phase can quickly filter out irrelevant traffic flows, leaving only suspicious ones for future analysis. The second phase inputs suspicious traffic and outputs confirmed cryptomining traffic. At last, the third detection phase utilizes an LSTM model to distinguish cryptojacking traffic from user-initiated cryptomining traffic.

3.4.1 Preprocessing.

CJ-Sniffer requires the timestamps and six fields in the IP packet headers for detection, which are the source and destination IP addresses, the source and destination port numbers, the protocol type, and the packet size. CJ-Sniffer is therefore content-agnostic, as it does not require any payload information.

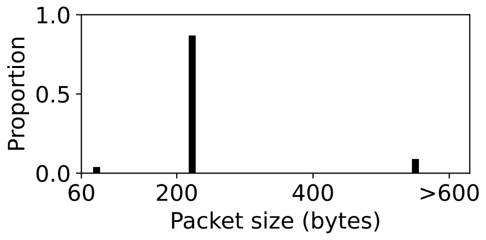
To obtain this content-agnostic data, we recommend installing sFlow [307] in the router or switch to stream traffic flows to CJ-Sniffer in real time. Other network traffic capturing engines like Netmap [325] and PF-RING [33] are also compatible with the proposed approaches. Once CJ-Sniffer fetches the traffic flows, it extracts the aforementioned data fields and stores them in a table for future analysis. In the table, each entry represents a received packet. Meanwhile, CJ-Sniffer discards all other information from the traffic flows.

3.4.2 Phase one: rapid filtration.

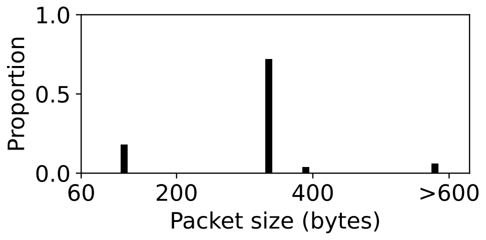
In phase one, CJ-Sniffer rapidly filters out irrelevant network traffic and picks out only suspicious traffic for future analysis. This step can significantly reduce the size of the traffic data for inference, increasing the throughput of CJ-Sniffer.

To conduct rapid filtration, CJ-Sniffer first eliminates packets without payload (e.g., TCP SYN, ACK, and FIN packets), packets of irrelevant protocols (e.g., ICMP), and internal packets. CJ-Sniffer then groups remaining packets by connections, which are defined as consecutive packets that are sent between the same IP addresses and port numbers. Then, CJ-Sniffer inspects the packet sizes in each connection to judge whether it could be a suspicious cryptomining connection.

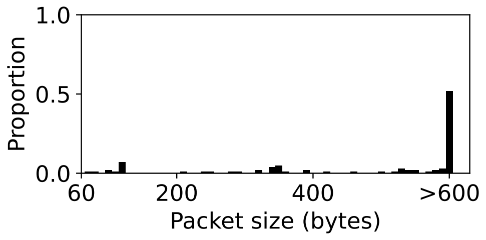
We define a sliding time window to monitor each connection and make a judgment. During each time window t, CJ-Sniffer collects a list of packets P ($P = \{p_1, p_2, p_3, ..., p_n\}$) from the ongoing connection. It then represents



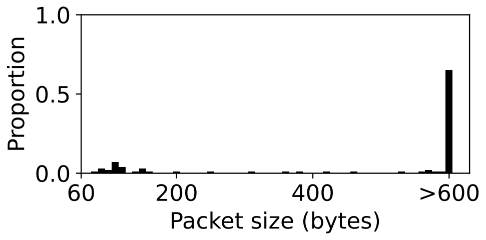
(a) Outbound packets of a cryptomining connection.



(b) Inbound packets of a cryptomining connection.



(c) Inbound packets of a Steam downloading connection. Most packets are larger than 600 bytes.



(d) Inbound packets of a video streaming connection. Most packets are larger than 600 bytes.

Figure 20. Packet size distribution of different types of traffic.

inbound and outbound packets with their size values and stores them into two lists (l_{in} for inbound packets and l_{out} outbound packets) respectively. Once this time window is about to end, CJ-sniffer will check the value distributions of l_{in} and l_{out} to determine whether the connection is suspicious. According to the traffic measurement study in Section 3.3.1, CJ-Sniffer utilizes three sets of rules to determine suspicious connections:

– The majority of the packets' sizes should lie within the cryptomining packet size range;

– The majority of the outbound packets' sizes should be uniform (illustrated in Figure 20a);

– The majority of the inbound packets' sizes should have values drawn from the same narrow interval (illustrated in Figure 20b).

Once packets within a connection follow these three rules, CJ-Sniffer will label this connection as suspicious and pass it onto the next phase for deeper analysis. Note, labeling a connection as suspicious does not mean this connection is confirmed to be related with cryptomining. For example, some connections generated by NTP or DNS can have similar distribution of packet sizes. Hence, CJ-Sniffer only filters out obviously irrelevant traffic in this phase to accelerate the detection process.

3.4.3 Phase two: detection of cryptomining. In phase two, CJ-Sniffer uses filtered network traffic from phase one as input and outputs detected cryptomining traffic. According to the cryptomining traffic study in Section 3.3.1, to accurately identify cryptomining traffic, CJ-Sniffer inspects the packet intervals of suspicious connections to determine whether they are generated by the cryptominer or the mining pool. Specifically, CJ-Sniffer compares both the inbound and outbound packet interval distributions to distributions from collected cryptomining traffic. As long as one of the inbound or outbound packet intervals follow the same distribution with cryptomining traffic data, CJ-Sniffer will label the connection as a cryptomining connection.

The distribution compliance test is conducted with the Two-Sample Kolmogorov–Smirnov (KS) test [264], which is a nonparametric testing approach to determine whether two data samples come from the same distribution. Compared with other approaches to testing the distribution compliance, the KS test has no restrictions on the size of data sample, which means little cryptomining traffic data can help achieve decent accuracy. Moreover, the KS test is distribution-free.

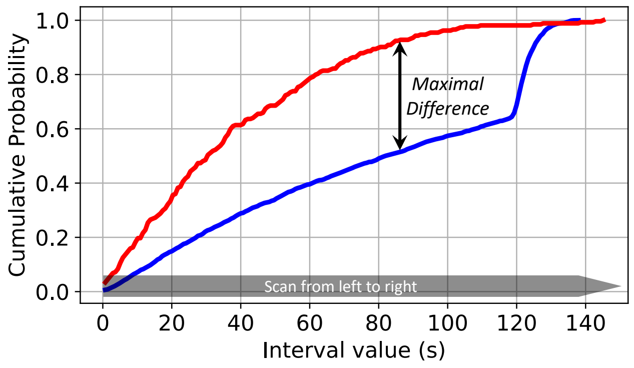


Figure 21. Illustration of the two-sample Kolmogorov–Smirnov statistic. The blue line corresponds to the empirical distribution function of labeled cryptomining traffic. The red line corresponds to the empirical distribution function of downloading traffic. The black arrow is the two-sample KS statistic.

Users can easily update the contrast sample to cover the latest cryptomining traffic regardless of the sample's distribution.

The Two-Sample KS test works as follows. Suppose that the inbound or outbound packet interval sample from P has size m with an observed cumulative distribution function of $F(x)$. Furthermore, suppose that the labeled cryptomining sample Q has size n with an observed cumulative distribution function of $G(x)$. CJ-Sniffer defines the null hypothesis (H_0) as: both samples come from a population with the same distribution. It also defines the Two-Sample KS statistic $D_{m,n}$ with Equation 3.1 (illustrated in Figure 21).

$$D_{m,n} = \max_x |F(x) - G(x)|. \tag{3.1}$$

After calculating $D_{m,n}$, CJ-Sniffer rejects the null hypothesis at significance level α if $D_{m,n} > D_{m,n,\alpha}$, where $D_{m,n,\alpha}$ is the critical value and can be calculated with Equation 3.2.

$$\begin{aligned} D_{m,n,\alpha} &= c(\alpha)\sqrt{\frac{n+m}{n \cdot m}} \\ &= \sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1+\frac{m}{n}}{2m}}. \end{aligned} \tag{3.2}$$

Conversely, if $D_{m,n} \leq D_{m,n,\alpha}$, CJ-Sniffer will accept the null hypothesis H_0 and label the incoming traffic as cryptomining.

The significance level α in the Two-Sample KS test is the probability of rejecting the null hypothesis when it is true. Users of CJ-Sniffer can adjust the value of α to reach different detection sensitivities. A large significance level α can lead to a small critical value $D_{m,n,\alpha}$, which will raise the standard of the distribution compliance test. In our implementation, we set α as 0.10, which is a relatively large value but can increase the usability of CJ-Sniffer by reducing the false positive rate.

Algorithm 1 demonstrates the detailed procedure that CJ-Sniffer utilizes to determine cryptomining traffic. CJ-Sniffer only tests the distribution compliance of inbound packet intervals, as inbound packet intervals are more robust compared with outbound packet intervals (discussed in Section 3.3). Moreover, to reduce the process time, the cumulative distribution function of labeled cryptomining traffic is calculated beforehand. Therefore, when receiving new suspicious traffic, CJ-Sniffer only needs to build one cumulative distribution function.

Once CJ-Sniffer completes the analysis in phase two, network operators can choose the next step according to their needs. For instance, if some companies and institutions prohibit any cryptomining inside their networks, then they can stop at this phase and block any connection that is labeled as cryptomining. Meanwhile, some network operators allow user-initiated cryptomining activities. Nonetheless, they still want to distinguish cryptojacking traffic to safeguard users' computing resources. In this case, CJ-Sniffer can enter phase three to dig further into the labeled cryptomining traffic.

Algorithm 1 Cryptomining traffic detection using KS test.

```
1: Input: $P$, $m$, $Q$, $G(x)$, $n$, $k$, $\alpha$    ▷ $P$ is the packet list with $m$ packets, $Q$ is the labeled cryptomining packet list with $n$ packets, $G(x)$ is the cumulative distribution function of $Q$, $k$ is the granularity for calculating the KS statistic, $\alpha$ is the significance level
2: Output: 1 for cryptomining traffic, 0 for other traffic
3: $l_P = inboundInterval(P)$    ▷ extract the inbound packet intervals and store them in a list
4: $l_Q = inboundInterval(Q)$
5: $r = max(l_Q) - min(l_Q)$    ▷ calculate the range of $G(x)$
6: initialize list $l_d$    ▷ to store the differences of two cumulative distribution functions (CDFs)
7: for $i$ in $range(k)$ do
8:     $x \longleftarrow \frac{i \cdot r}{k} + min(l_Q)$
9:     $l \longleftarrow \{j | j \in l_P \ and \ j \leq x\}$
10:     $f \longleftarrow \frac{l.size()}{l_P.size()}$    ▷ calculate the CDF value at $x$ for $P$
11:     append $|f - G(x)|$ to $l_d$
12:     if $f == 1$ then
13:         break
14:     end if
15: end for
16: $D_{m,n} \longleftarrow max(l_d)$
17: if $D_{m,n} \leq \sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1+\frac{m}{n}}{2m}}$ then
18:     return 1    ▷ accept the hypothesis $H_0$
19: else
20:     return 0
21: end if
```

3.4.4 Phase three: detection of cryptojacking.

In the third phase, CJ-Sniffer uses detected cryptomining traffic as input and outputs identified cryptojacking traffic. According to the detection results of CJ-Sniffer, network operators can conduct access control only on cryptojacking connections while still leaving user-initiated cryptomining connections alive.

Enlightened from the measurement results in Section 3.3.2, CJ-Sniffer distinguishes cryptojacking traffic from user-initiated cryptomining traffic by inspecting the long-term robustness of the result submission messages. For various reasons, the hash rate of cryptojacking activities is relatively unstable compared with user-initiated cryptomining. This hash rate instability further affects the frequency of result submission messages (msg_r). The key to identifying cryptomining connections is to recognize such frequency instability.

To achieve the goal, CJ-Sniffer utilizes a LSTM machine learning model to learn the cryptojacking traffic patterns, with cryptojacking traffic as positive samples and user-initiated cryptomining traffic as negative samples. Then, CJ-Sniffer applies the trained LSTM model to identify cryptojacking traffic. LSTM is a type of artificial recurrent neural network (RNN) architecture used in the field of deep learning [200]. Compared with other candidate approaches, LSTM is particularly suitable to detect cryptomining traffic for the following reasons: (1) as an RNN variant, LSTM is good at processing time series or sequential data, such as cryptomining traffic; (2) LSTM introduces both a short-term and a long-term memory component, allowing it to uncover hidden frequency changes out of traffic data from a long-term perspective; (3) LSTM is a learning-based approach, which means it can automatically and dynamically learn the detection thresholds from our collect dataset without any human interventions. Nonetheless, the detection

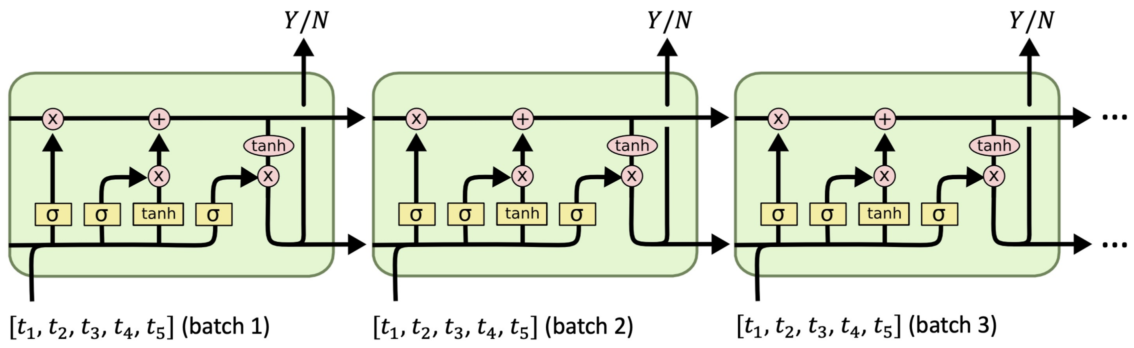

Y/N
Y/N
Y/N
σ
σ
tanh
σ
tanh
$[t_1, t_2, t_3, t_4, t_5]$ (batch 1)
$[t_1, t_2, t_3, t_4, t_5]$ (batch 2)
$[t_1, t_2, t_3, t_4, t_5]$ (batch 3)

Figure 22. Structure of the LSTM model that CJ-Sniffer utilizes.

component here is modular. Users may use other machine learning algorithms or statistical approaches to fit their environments once the inputs are the same.

As for the input data, CJ-Sniffer extracts the variation vector v from outbound traffic to profile the changes in result submission message frequency. CJ-Sniffer first picks out only the outbound packets from the traffic and divides them into batches. Every batch consists of six consecutive packets with five consecutive packet intervals. Then, CJ-Sniffer generates a variation vector v to represent each batch of data, where $v = [t_1, t_2, t_3, t_4, t_5]$ and t_n denotes the interval of two consecutive packets. CJ-Sniffer will input the variation vectors into the LSTM model and conduct cryptojacking detection batch by batch.

Figure 22 illustrates the structure of the LSTM model that CJ-Sniffer utilizes. As the input data is not complicated, CJ-Sniffer employees a Vanilla LSTM model, which has a single hidden layer of LSTM units, and an output layer used to make the decision. We set the number of neurons in the hidden layer to 20 according to a rule of thumb that was introduced in [125]. Equation 3.3 demonstrates the rule, where N_i denotes the number of input neurons, N_o denotes the number of output neurons, N_s denotes the number of samples in the training set, α is an arbitrary scaling factor (usually ranges from 2 to 10), and N_h denotes the maximum number of neurons in the hidden layer.

$$N_h = \frac{N_s}{\alpha \cdot (N_i + N_o)}. \tag{3.3}$$

Moreover, the LSTM uses binary cross entropy as the loss function and Sigmoid as the activation function for the output neuron, as this combination is the most commonly used for binary classification problems. CJ-Sniffer keeps inputting variation vectors as time-series data to the LSTM, until the incoming traffic is identified as cryptojacking or all the batches have been processed.

The LSTM model requires training before it can be used. Since there are no existing cryptojacking traffic datasets available in public repositories, we captured both user-initiated cryptomining traffic and cryptojacking traffic to train the LSTM model. We also release part of the packet-level dataset with this paper.

3.5 Evaluation

We evaluated CJ-Sniffer in campus network environments with real-world network traffic. In this section, we first describe how we collect labeled cryptomining traffic and real-world contrast traffic (Section 3.5.1). Then, we show our evaluation results of CJ-Sniffer regarding the efficacy of cryptomining traffic detection (Section 3.5.2), the ability of detecting other cryptocurrencies' mining traffic (Section 3.5.3), the efficacy of cryptojacking traffic detection (Section 3.5.4), and comparisons with other network-based approaches (Section 3.5.5). In the end, we evaluate the operation efficiency and the real-world deployability of CJ-Sniffer (Section 3.5.6.2).

3.5.1 Data collection. To support the measurement study of cryptomining traffic, train the statistical model and the machine learning model, and promote related research, we collected a labeled cryptomining traffic dataset from multiple calculation platforms, including both servers and personal computers,

Table 13. Information of the collected cryptomining dataset.

Cryptocurrencies	**Size**	**Length**	**Mining Chips**
XMR, ETH	250 MB	$\sim$ 750 hours	Intel Core i5-5257U, Intel Core i7-6820HQ, Apple M1, AMD Ryzen 5 1400 quad-core, NVIDIA GeForce GTX 1080, Intel Core i7-8700K, Intel Xeon CPU E5-2430, AMD Radeon RX 570, AMD Ryzen 5 1400 quad-core + AMD Radeon RX 570.

Table 14. Information of the collected contrast network traffic.

Network	**Size**	**Length**	**Types of Traffic Contained**
Campus (10 Gbps), Lab (1 Gbps)	316 GB	$\sim$ 35,100 connections, $\sim$ 2,000 hours in total	**Traffic without obvious periodic regularities:** Media streaming, VoIP service, HTTP, SMTP, FTP, Skype, Zoom, Gaming, Tunneling and proxy services, etc. **Traffic with obvious periodic regularities:** NTP, DNS, Control services of IoT appliances, Notification services, STUN, Google static content, etc.

CPUs and GPUs. The cryptomining traffic dataset contains both user-initiated cryptomining traffic and cryptojacking traffic.

To collect the dataset, we installed both user-initiated cryptomining and cryptojacking software on different computing devices in different network environments (e.g.home, lab, campus, etc.). We also created several websites with cryptojacking scripts from WebMinePool [28], CoinIMP [19], Easy Pool Miner [21], and Minero [25]. Then, we used Wireshark [291] to capture the packet-level traffic along with labeling information generated by these cryptomining activities. Table 13 shows the basic information of the collected dataset. The development and evaluation of CJ-Sniffer is based on this dataset. Moreover, we release a subset of the dataset to public [147], which has around 550 hours of cryptomining traffic, with all the noise and human-behavior-related traffic removed due to ethical considerations.

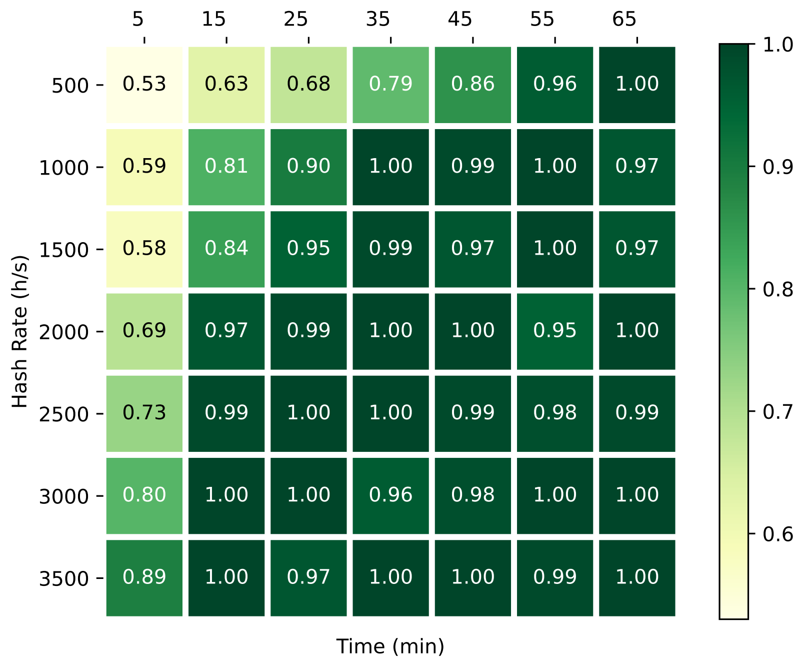


Figure 23. Accuracy scores of detecting cryptomining traffic with different hash rates and length.

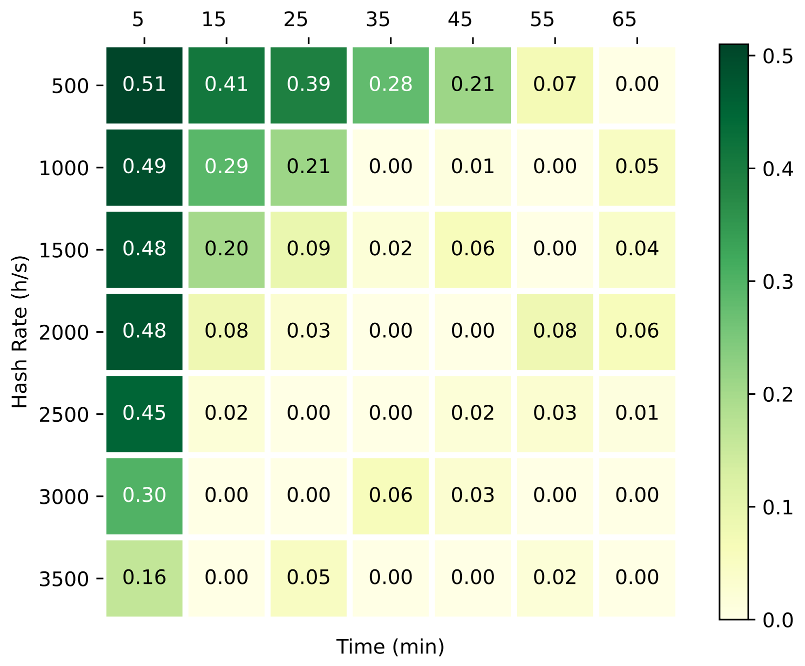


Figure 24. False negative rates of detecting cryptomining traffic with different hash rates and length.

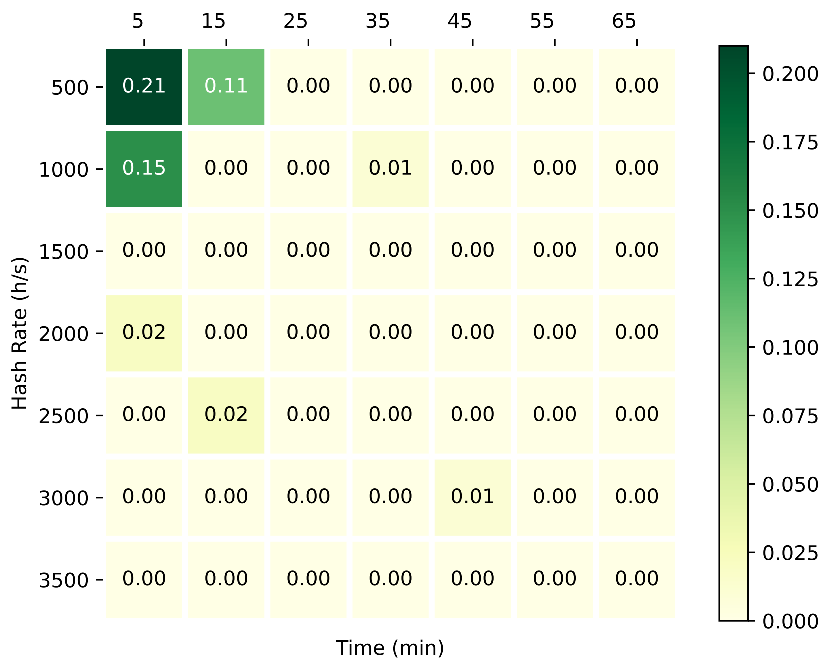


Figure 25. False positive rates of detecting cryptomining traffic with different hash rates and length.

Besides, we captured around 316 GB of contrast traffic from our lab network (link bandwidth: 1 Gbps) and campus network (link bandwidth: 10 Gbps) [1]. Table 14 shows the basic information of the collected contrast traffic. By using port-based and graphlet-based [225] traffic classification approaches, we classified the contrast network traffic into several categories (e.g.HTTP, NTP, SMTP, etc.). Furthermore, according to the traffic patterns, we divided these categories into two groups. One is traffic without obvious periodic regularities, which is easier to distinguish from cryptomining traffic. The other is traffic with obvious periodic regularities, which is more difficult to distinguish from cryptomining traffic. The collected contrast traffic basically cover most types of network traffic we can see

[1]We anonymously collected the contrast traffic and omitted all the private content data before storage. We have also obtained the Institutional Review Board (IRB) approval for the traffic collection.

from medium-sized companies/institutions. We later mixed all the contrast traffic with the cryptomining traffic to evaluate the detection accuracy of CJ-Sniffer.

3.5.2 Efficacy of cryptomining traffic detection. To evaluate CJ-Sniffer's efficacy of detecting general cryptomining traffic, we divided the dataset into two parts—the training set and testing set. The training set is used to train the detection model, containing around 200 hours of labeled cryptomining traffic. The testing set is treated as the input of CJ-Sniffer to evaluate its efficacy. As cryptomining traffic and other types of traffic are unbalanced in reality, the ratio of contrast traffic to cryptomining traffic is more than 20:1 in our testing set. Besides, to evaluate the detection accuracy in different scenarios, we further divided the testing set into seven groups according to the cryptomining hash rates.

Figure 23 demonstrates the accuracy scores of detecting cryptomining connections using CJ-Sniffer. Figure 24 demonstrates the false negative rates of detecting cryptomining traffic using CJ-Sniffer; We can see that detecting cryptomining traffic of different hash rates or lengths will derive totally different accuracy and false negative scores. As a statistics-based cryptomining detection approach, CJ-Sniffer can achieve a better efficacy with a larger data sample. Under this scenario, cryptomining traffic with longer durations or larger hash rates is easier to be detected by CJ-Sniffer, since these sets of traffic contain more interval samples for analysis. In general, to reach an accuracy of more than 0.95 in detecting both low-hash-rate and high-hash-rate cryptomining traffic, CJ-Sniffer needs to collect around 160 network packets generated from the device in each processing unit. If the cryptojacked devices are all high-performance devices (with hash rates of more than 2000h/s), they only need 15 minutes to generate this many

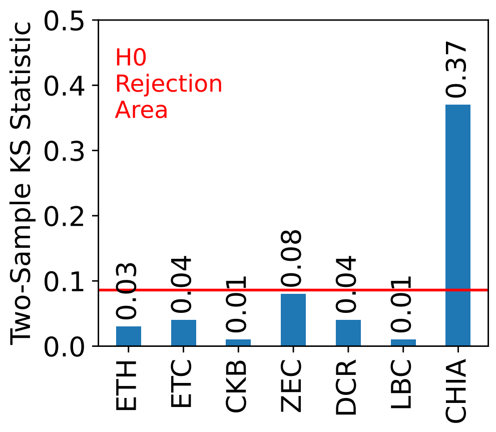


Figure 26. Two-sample KS statistics between XMR and other cryptocurrencies' mining traffic.

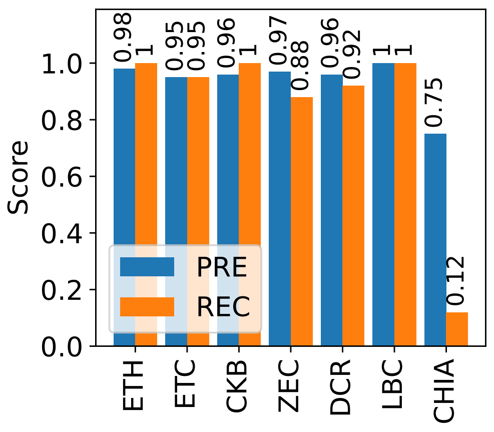


Figure 27. CJ-Sniffer's precision and recall scores in identifying other cryptocurrencies' mining traffic.

network packets, which is significantly quicker compared with MineHunter that needs around 2 hours of traffic to achieve a similar efficacy.

Figure 24 demonstrates the false negative rates of detecting cryptomining traffic using CJ-Sniffer. Similar to accuracy scores, CJ-Sniffer needs shorter time to detect high-hash-rate devices' cryptomining traffic. Once CJ-Sniffer collects 55 minutes of traffic in each processing unit, it can achieve zero false negative rates for detecting both low-hash-rate and high-hash-rate cryptomining traffic.

3.5.3 Adaptability to other cryptocurrencies. CJ-Sniffer is mainly built upon XMR mining data. However, cryptomining/cryptojacking traffic may be associated with other cryptocurrencies. In this section, we examine the adaptability of CJ-Sniffer. We collected several hours of cryptomining traffic of other cryptocurrencies (i.e.ETH, ETC, BTC, DCR, LBC, ZEC, and CHIA). The data volume may be insufficient for thorough measurement but is enough for testing. We then evaluated the ability of CJ-Sniffer (trained with XMR traffic) in detecting such cryptomining traffic.

Table 15. Efficacy of cryptojacking traffic detection with different system loads of background processes.

	System load of background processes.				
	0%	10%	20%	30%	40%
Accuracy	0.4714	0.5467	0.6176	0.7244	0.9390
Precision	0.8750	0.9130	0.9231	0.9348	0.9770
Recall	0.0875	0.2413	0.3673	0.5181	0.9140
FPR	0.0167	0.0317	0.0417	0.0411	0.0282
FNR	0.9125	0.7586	0.6327	0.4819	0.0860
F1 Score	0.1591	0.3818	0.5255	0.6667	0.9444
	System load of background processes.				
	50%	70%	80%	90%	100%
Accuracy	0.9630	0.9877	0.9877	0.9938	0.9890
Precision	0.9770	0.9888	0.9891	0.9890	0.9897
Recall	0.9551	0.9888	0.9891	1	0.9897
FPR	0.0274	0.0135	0.0143	0.0143	0.0118
FNR	0.0449	0.0112	0.0109	0.0000	0.0103
F1 Score	0.9659	0.9888	0.9891	0.9945	0.9897

The test is conducted on normalized traffic data. Figure 26 demonstrates the two-sample KS statistics between XMR and other cryptocurrencies' cryptomining packet intervals. The closer these values are to 0, the more similar these traffic intervals are to XMR's, and the more likely CJ-Sniffer can detect such traffic. From the results, we can see that cryptomining packet intervals of ETH, ETC, BTC, DCR, LBC, and ZEC come from the same distribution as XMR's, because their KS test statistics are too small to reject the H_0 hypothesis. Conversely, CHIA, possibly due to the adoption of a different consensus mechanism (i.e.Proof of Space), is quite different from XMR regarding cryptomining traffic. The detection evaluation is consistent with the statistics (Figure 27). CJ-Sniffer achieves decent precision and recall scores in detecting almost all the PoW-based cryptomining traffic.

3.5.4 Efficacy of cryptojacking traffic detection. CJ-Sniffer observes the hash rate robustness of ongoing cryptomining traffic to distinguish cryptojacking from user-initiated cryptomining. Therefore, the efficacy of cryptojacking traffic detection may be influenced by the background process running on the device. Here, the background process refers to all the computing processes executed by the legitimate user of the cryptojacked device. Theoretically, if the background processes cause a high system load, the computing resource allocated to cryptojacking programs will be volatile, leading to obviously unstable hash rates and becoming easier to be detected by CJ-Sniffer. However, when the background processes only cause a little system load, the system is more likely to allocate stable computing resources to cryptojacking programs, even if they are in low priorities. Therefore, CJ-Sniffer may fail to detect such cryptojacking activities.

We evaluate the efficacies of cryptojacking traffic detection with different background processes and present the results in Table 15. From the table, we can see that CJ-Sniffer can achieve good accuracy scores and false positive rates in any situation. However, we also notice that it can only reach decent recall scores and false negative rates when the loads of background processes are higher than 30%. Otherwise, CJ-Sniffer may have difficulty distinguishing cryptojacking traffic from user-initiated cryptomining traffic. Fortunately, in these scenarios, the computing device is in idle time, thereby limiting the effect such cryptojacking activities have on legitimate processes. Therefore, a cryptojacking attack is less of a concern under such circumstances. Still, CJ-Sniffer can tell network administrators that these traffic are cryptomining traffic in all the cases.

3.5.5 Comparison evaluation. In this subsection, we compare CJ-Sniffer with four other representative solution: 1. Cisco's commercial solution [45]

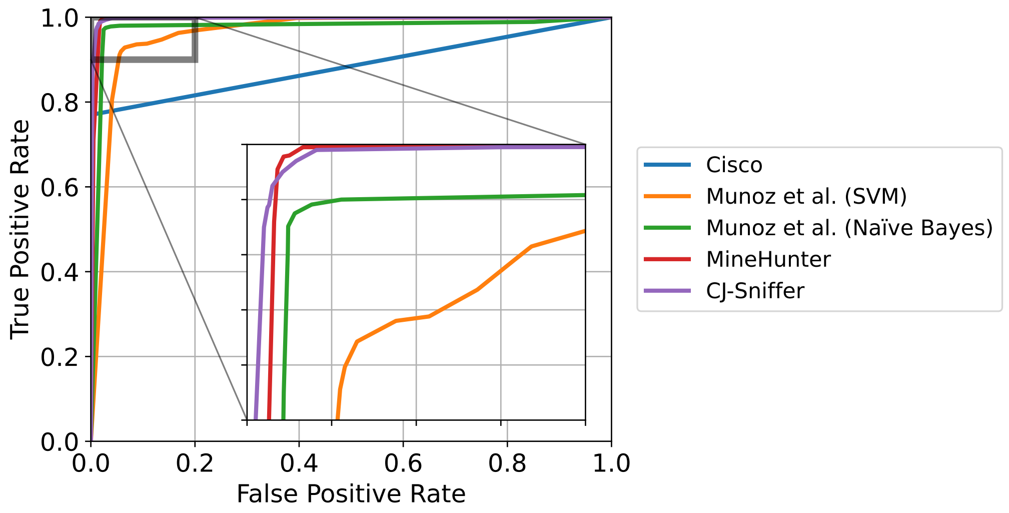


Figure 28. ROC curves of selected approaches for cryptomining traffic detection.

that integrated in routers; 2. SVM-based approach proposed by Munoz et al. [210]; 3. Naïve Bayes-based approach proposed by Munoz et al. [210]; 4. MineHunter [443].

3.5.5.1 Comparison of cryptomining traffic detection. We use similar evaluation approaches in Section 3.5.2 to conduct the comparison evaluations for cryptomining traffic detection. The only difference is that each testing unit contains around 2 hours of traffic, so approaches that require a large amount of traffic (i.e.MineHunter and Munoz et al.) can achieve the best efficacy. Figure 28 illustrates receiver operating characteristic (ROC) curves generated by the five approaches. We can see that CJ-Sniffer and MineHunter perform obviously better than approaches proposed by Munoz et al. and Cisco. CJ-Sniffer and MineHunter achieve similar accuracies, while CJ-Sniffer is better in false-positive rates, and MineHunter is better in true-positive rates.

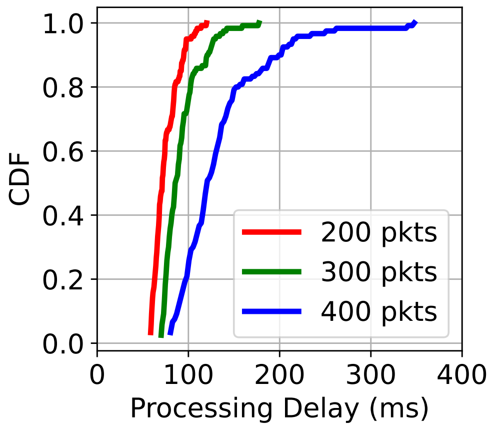
As for the processing delay, all these five approaches can output the detection results within 2 seconds. Therefore, all of them can meet the velocity requirements in cryptomining traffic detection.

3.5.5.2 Comparison of cryptojacking traffic detection. Among the five approaches, CJ-Sniffer is the only one that can distinguish cryptojacking traffic from user-initiated cryptomining traffic. Other approaches simply treat these two groups of traffic as the same. Hence, CJ-Sniffer allows network operators to conduct more elastic and flexible security managements. For example, a network administrator can only filter cryptojacking traffic while preserving user-initiated cryptomining traffic from a device, which is a common requirement for many Virtual Private Servers (VPS) providers. For networks with strict usage restrictions (e.g.campus networks, enterprise networks, etc.), network administrators can simply utilize the first two phases of CJ-Sniffer to identify all cryptomining traffic.

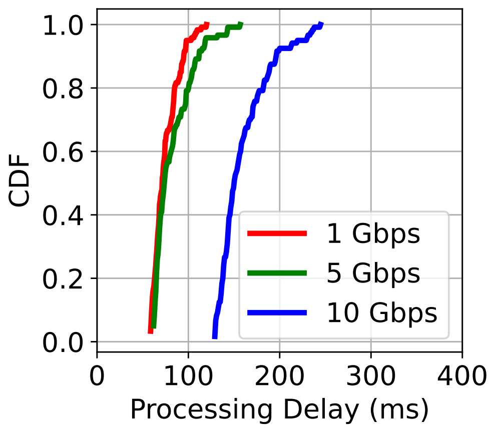
3.5.6 System Efficiency. In this section, we evaluate the efficiency of CJ-Sniffer by time complexity analysis and measuring the system's processing delay in real-world environments.

3.5.6.1 Time complexity. Assume the total number of input packets is n. The time complexity of the *preprocessing* module is $O(n)$, as CJ-Sniffer simply receives network traffic from traffic capture engines and extracts necessary attributes from each packet header.

Then, CJ-Sniffer moves to *phase one*. CJ-Sniffer goes through each packet's extracted information to filter out noise and group them by connections, which takes $O(n)$ time to complete. After that, assume there are an $(0 \leq a \leq 1)$ packets left, CJ-Sniffer still needs $O(an)$ time to complete rapid filtration. As a result, CJ-Sniffer's time complexity for phase one is $O(n + an)$.



(a) Cumulative distribution functions (CDFs) of processing delays with different numbers of packets in each processing unit (link bandwidth: 1 Gbps).



(b) Cumulative distribution functions (CDFs) of processing delays when monitoring links with different bandwidths (processing unit: 200 pkts).

Figure 29. Processing delays of CJ-Sniffer under different operation circumstances (different link bandwidths & sizes of processing units).

In *phase two*, assume there are bn ($0 \leq b \leq 1$) packets left. Since the labeled samples' cumulative distribution function $G(x)$ is pre-built, CJ-Sniffer can utilize Algorithm 1 to detect cryptomining traffic, allowing it to complete the analysis in $O(kbn)$ time.

In *phase three*, CJ-Sniffer employees an LSTM model to detect cryptojacking traffic. The LSTM model performs detections batch-by-batch and needs $O(1)$ time to process each batch. Assumes there are cn ($0 \leq c \leq 1$) batches, CJ-Sniffer needs $O(cn)$ to complete phase three.

Overall, CJ-Sniffer takes $O((a + kb + c)n)$ time to detect cryptojacking traffic from captured network traffic, as it conduct all the detection processes sequentially. In addition, k is a constant and a, b, c are numbers between zero and one. Therefore, CJ-Sniffer's time complexity is $O(n)$, where n is the total number of input packets.

3.5.6.2 Processing delay. We then measured the processing delay of CJ-Sniffer in the different real-world environments. Here, the processing delay refers to the time it takes for CJ-Sniffer to output the detection result after the data collection is completed. According to the design of CJ-Sniffer, there are two factors that can affect the processing delay: (1) the number of packets in each processing unit, and (2) the bandwidth of the link that CJ-Sniffer is monitoring. We changed both of the factors to evaluate the efficiency of CJ-Sniffer. We used a personal computer with an i7 8700k 4.7-GHz CPU and 32-GB memory to conduct the evaluation. The network traffic was ported from the gateway routers of our campus network and lab network.

Figure 29a demonstrates the delay with different number of packets in each processing unit. CJ-Sniffer takes 89 milliseconds on average to output the result when there are 200 packets per unit. If the number increases to 300, the processing delay just slightly increases to around 100 milliseconds. Although there is a noticeable performance drop when the number of packets in each processing unit reaches 400, according to evaluations in Section 3.5.2, 200 packets per detection unit is enough to reach a deployable accuracy.

Figure 29b demonstrates the delay with different link bandwidths. We can see that with a personal computer's computing power, CJ-Sniffer is able to monitor a campus-level or medium-company-level network (10 Gbps) and finish processing the data in less than 0.25 seconds.

Therefore, the processing time data from the real deployment of CJ-Sniffer supports the aforementioned time complexity analysis result. CJ-Sniffer's monitoring and detection can reach line speed for most companies, campuses, or institutes, with affordable computing power.

3.6 Discussion

In this section, we discuss the limitations of CJ-Sniffer and its generalizability to other FGTA applications.

3.6.1 Limitation.

A primary contribution of CJ-Sniffer is to use content-agnostic data to detect cryptomining traffic and further distinguish cryptojacking traffic from user-initiated cryptomining traffic in the network, which can protect users' privacy and increase the efficiency during the detection process. However, CJ-Sniffer still has a few open issues. In this subsection, we discuss these open issues, indicating possible limitations of CJ-Sniffer and raising several avenues for future research.

3.6.1.1 Bypassing CJ-Sniffer.

CJ-Sniffer is fundamentally a detection approach grounded in traffic analysis. As elaborated upon in Section 2.6, a myriad of countermeasures aimed at circumventing traffic analysis have been put forth by researchers and developers over the past two decades. These include strategies such as packet padding [222], dummy packets [404], and traffic morphing [418], among others. While some of these countermeasures may be effective in bypassing CJ-Sniffer, others may not. We explore the potential for utilizing these countermeasures to circumvent CJ-Sniffer in the subsequent discussion.

1. **Packet Padding or Adding Dummy Packets:** Cryptojackers are unable to directly alter their cryptomining traffic through padding or the addition of dummy packets to elude our detection system. This is because present-day cryptomining activities are largely dependent on public mining pools, which necessitate miners to adhere strictly to public mining protocols (i.e., Stratum [319]). These protocols demand the removal of superfluous

content in packets to prevent virus injection. Therefore, any modification to cryptomining traffic could potentially lead to disconnections from the mining pool.

2. **Packet Timing Manipulation:** In theory, cryptojackers could manipulate the timing of their cryptojacking traffic to bypass CJ-Sniffer, rendering the traffic similar to user-initiated cryptomining traffic or even normal traffic. Even though this strategy could potentially achieve their goal, it would significantly diminish the cryptojackers' profits. Cryptomining is a highly competitive and time-sensitive process, and any delay in the delivery of mining packets might lead to other miners submitting results first, resulting in the rejection of the delayed results. Conversely, if the mining packets are sent prematurely, the mining device has not computed the correct results yet, which will also lead to rejection. Therefore, this strategy is rarely utilized by cryptojackers to bypass FGTA-based detection systems.

3. **Use of Private Mining Pool with Unknown Protocols:** The possibility of bypassing CJ-Sniffer by establishing a private mining pool with an unknown cryptomining protocol is also unlikely. Given the intense computational power competition [106] among mining pools, the hash rates of private mining pools are considerably lower than those of popular public mining pools. This makes private mining pools inefficient for mining new coins. Thus, to mine cryptocurrencies efficiently, attackers rarely resort to private mining pools.

4. **Traffic Proxy or Tunneling:** Nonetheless, it is feasible for cryptojackers to exploit third parties to help conceal their cryptomining traffic (e.g., proxy,

VPN, Tor with traffic obfuscation [148]), thereby bypassing CJ-Sniffer. While we have yet to observe this defensive design in existing cryptojacking software, studying the detection of cryptojacking traffic in the presence of traffic tunneling presents an exciting direction for future research.

3.6.1.2 Adaptability to other network environments. CJ-Sniffer's adaptability in different network environments can significantly affect its usability because re-training the detection model is a time-consuming process for learning-based approaches. Fortunately, according to our collected data in different network environments and investigations of the cryptomining mechanism, changes in cryptomining traffic are independent of different network environments. Therefore, a cryptomining detection model trained in one network environment can be directly applied to another network environment if the input traffic is of the same format (e.g., sFlow, in our implementation). However, blockchain systems may evolve by upgrading their mining mechanisms, such as the Arrow Glacier upgrade of ETH [32] and the CryptoNight V7 upgrade of XMR [123]. These upgrades can bring changes to the cryptomining traffic patterns, thereby making detection models trained by previous data no longer effective. To tackle this issue, researchers need to continue enriching and updating the cryptomining traffic dataset so that the dataset can cover most recent cryptomining traffic patterns.

3.6.1.3 Direct hash rate inference. CJ-Sniffer detects cryptojacking by identifying hash rate fluctuations. A more straightforward method is to directly identify the specific hash rates from the traffic, which not only can help detect cryptojacking, but also can estimate the amount of computing power that is devoted to cryptomining. However, identifying specific hash rates is more challenging than identifying hash rate fluctuations, especially when there is no content data involved. In the future, extending CJ-Sniffer with more traffic features, preprocessing steps, or training data so that it can discover more information from content-agnostic traffic data is a promising research direction.

3.6.2 Approach generalization.

CJ-Sniffer is specifically designed for detecting cryptojacking traffic. The specificity of the training data and the detection model makes it difficult to generalize CJ-Sniffer to other FGTA applications. However, the traffic processing pipeline, methodologies, and techniques used in CJ-Sniffer may be applied to other FGTA applications. Here, we list several designs of CJ-Sniffer that may be useful for other FGTA applications.

3.6.2.1 Two phases of traffic processing.

CJ-Sniffer employs a two-phase traffic processing pipeline designed to boost the speed of traffic processing. Initially, it employs straightforward yet efficient rules to discard obviously irrelevant traffic, such rules may include criteria like port number, packet size, among others. Following this, CJ-Sniffer employs a more complex but precise detection model to discern target traffic from the remaining data pool. This pipeline, thanks to its improved detection efficiency, could be beneficially applied to other FGTA applications, thus rendering them more viable for scenarios involving high traffic throughput.

3.6.3 Rapid KS test for packet interval distribution.

CJ-Sniffer capitalizes on the KS test, applied to the packet interval distribution, for the detection of cryptomining traffic. Given that the packet interval distribution can be considered as a unique fingerprint for numerous web applications, our proposed method holds potential applicability for other TA tasks related to application identification. Moreover, we have introduced a rapid KS test methodology capable of significantly diminishing the time complexity of the KS test to $O(n)$, as illustrated in Algorithm 1. This approach could also find extensive utilization in various other domains.

3.6.4 LSTM for time series traffic analysis. For the purpose of fine-grained traffic differentiation, CJ-Sniffer utilizes an LSTM model to scrutinize time series traffic data, thereby identifying the subtle fluctuations in the hash rate. The observed outcomes underscore the powerful efficacy of LSTM models in performing such tasks. This technique holds potential applicability for other FGTA applications that necessitate detailed traffic differentiation, which may include application-layer anomaly detection, QoE investigations, or the online social bot detection problem, as discussed in Chapter IV.

3.7 Conclusion

In this chapter, we extend the use case of FGTA to detect malicious activities that traditional TA-based methods found challenging to identify. In particular, we apply FGTA to detect cryptojacking traffic, harnessing the fine-grained traffic discrimination capabilities of FGTA to achieve a fine-grained and precise detection of such malicious traffic.

We propose CJ-Sniffer, a privacy-aware cryptojacking detection approach that only relies on content-agnostic network traffic to conduct detections. CJ-Sniffer applies a three-phase procedure to identify cryptojacking traffic. It first filters out obviously irrelevant traffic to increase the detection throughput. It then accurately detects cryptomining traffic by conducting distribution compliance tests on packet intervals. Lastly, CJ-Sniffer digs deeper into the packet interval patterns, utilizing an LSTM model to distinguish cryptojacking traffic from user-initiated cryptomining traffic.

By introducing traffic analysis into cryptojacking detection, CJ-Sniffer is able to safeguard computing resources with superior efficiency and deployability. With a personal computer and traffic flow access to the network gateway, CJ-Sniffer is able to provide real-time traffic monitoring for a company-level network. More importantly, the privacy of users will not be violated throughout the detection process. In addition, unlike other network-based solutions that simply treat all types of cryptomining activities the same, CJ-Sniffer can distinguish cryptojacking traffic from user-initiated cryptomining traffic through their subtle differences. Therefore, CJ-Sniffer can provide hierarchical detection results, allowing network operators to conduct more elastic security management.

We have evaluated CJ-Sniffer with real-world network traffic and found that its efficacy and efficiency are both high. With the computing power of an i7 8700k 4.7-GHz CPU and 32-GB memory, CJ-Sniffer is able to achieve an accuracy of over 99% for PoW-based cryptocurrency systems with reasonable delays when monitoring a campus-level network.

CHAPTER IV
LEARNING-BASED, CONTENT-AGNOSTIC DETECTION OF OSN BOT TRAFFIC

Having successfully utilized FGTA for fine-grained detection of cryptojacking traffic in the previous chapter, we persist in this chapter to expand the scope of FGTA's application. We aim to identify malicious activities that were previously hard to detect with conventional TA-based approaches, thereby demonstrating FGTA's capability for fine-grained traffic discrimination and its versatility. In particular, this chapter focuses on a more complex application-layer malicious activity detection challenge—differentiating OSN bots from genuine OSN users using FGTA. This chapter proposes a method called BotFlowMon that relies only on content-agnostic flow-level data as input to identify OSN bot traffic. To achieve the goal, BotFlowMon introduces several new FGTA algorithms and techniques, including aggregating network flow records to obtain OSN transaction data, fusing transaction data to extract features and visualize flows, and an innovative density-valley-based clustering algorithm to subdivide each transaction into individual actions.

This chapter is derived in part from the following published articles:

– Published as Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. "Towards learning-based, content-agnostic detection of social bot traffic." *IEEE Transactions on Dependable and Secure Computing 18, no. 5 (2020): 2149-2163* [152].

– Published as Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu. "BotFlowMon: Learning-based, content-agnostic identification of social bot traffic flows." *In Proceedings of 2019 IEEE Conference on Communications and Network Security (CNS), pp. 169-177*, 2019 [151].

– Published as Yebo Feng. "BotFlowMon: Identify Social Bot Traffic With NetFlow and Machine Learning." *University of Oregon, Master Thesis*, 2018 [145].

I am the leading author of the above articles. Most content of this chapter was written by me, and I was responsible for conducting all the system designs, implementations, evaluations, and the presented analyses.

4.1 Introduction

The past decades have witnessed a rapid expansion of online social networks (OSN). Facebook has achieved more than 2.196 billion active users around the world and Twitter has reached 336 million users [214]. Unfortunately, OSNs are increasingly threatened by software-controlled social bots [160] that impersonate real OSN users for troublesome or malicious purposes [440]. Even though not all social bots are malicious, as many are used for customer service and information dissemination, various attacks, abuses, and manipulations are based on social bots [159], such as infiltrating Twitter [73], launching spam campaigns [170], and performing financial fraud [23].

Existing approaches to detecting social bots need to utilize the social relationship structure or private content data from users' accounts, all of which can lead to privacy infringement and can only be executed by OSN providers. In this paper, we propose a new, *content-agnostic* social bot detection method called BotFlowMon. It takes network traffic flow information as input, which is NetFlow records [102] in this paper, to differentiate the social bot traffic from the real user traffic. As Internet/network service providers can easily collect NetFlow records, it is thus also convenient for them to deploy BotFlowMon, making social bot detection no longer dominated by OSN providers. Moreover, as NetFlow records are coarse-grained summaries of packet headers and contain *no* OSN content data from packet payload [354], BotFlowMon is also privacy-preserving.

BotFlowMon can detect social bot traffic accurately and quickly. It harnesses the power of machine learning on big data for the best efficacy in labeling social bot traffic versus real user traffic. BotFlowMon employs five modules:

1. The *preprocessing module* that filters out noises and irrelevant data from raw NetFlow records and extracts OSN-related traffic flows;

2. The *flow aggregation module* that transfers the NetFlow records into transaction-level datasets, making the characteristics of social bots more apparent for detection;

3. The *transaction fingerprint generation module* that, with a new data fusion technique, extracts features from transaction-level datasets, normalizes the values, and visualizes the flows;

4. The *transaction subdivision module* that employs a new, density-valley-based clustering algorithm to further divide each transaction into multiple actions, thus reducing training data volume and accelerating training;

5. The *machine learning & classification module* that uses the action-level data to construct a transaction-level social bot classification model with convolutional neural network (CNN) and multilayer perceptron (MLP).

BotFlowMon is the first work that leverages content-agnostic traffic flows to detect social bots. By embracing a suite of newly introduced techniques, BotFlowMon provides social bot detection at behavior level, which is a finer granularity than existing approaches; it can identify whether an individual OSN

Table 16. Comparisons of different social bot detection approaches.

Category	**Representative work**	**Primary input**	**Detection granularity**	**Timeliness**	**Privacy preservation**
Content-based	[101, 371, 429, 392], etc	OSN content	behavior, account (cluster)	mostly real time	No
Structure-based	[85, 179, 215, 388], etc	OSN topology	account	with the growth of topology	No
Crowdsourcing-based	[393, 128, 394, 316], etc	human judgment	account	with considerable delay	No
Traffic-flow-based	**BotFlowMon**	*network flows*	*behavior*	*real time*	*Yes*

behavior is from a real user or a social bot account. It also can detect social bots in real time; as soon as it receives network traffic, BotFlowMon can start detection immediately.

Our evaluation with 535 gigabytes (GB) of raw NetFlow data from a large university environment shows that BotFlowMon can identify social bot traffic with an accuracy of 96.1% (or around 90% if the user adjusts the false positive rate to zero). Simultaneously, the conciseness of NetFlow data provides an advantage for fast and efficient data processing. Even only running on a laptop with 2.7-Ghz CPU and 16-GB memory, BotFlowMon can support real-time social bot detection for a campus-level network; after the NetFlow records of a social bot are collected, it takes only 0.71 seconds on average to detect the social bot.

The rest of this paper is organized as follows. After we outline related work in Section 4.2, we describe BotFlowMon's design in Section 4.3 and evaluate it in Section 4.4. We discuss its limitations, open issues, and potential generalizations in Section 4.5 and conclude the paper in Section 4.6.

4.2 Related Work

As BotFlowMon is a content-agnostic approach to detecting social bots from network traffic, in this section we investigate other social bot detection approaches and content-agnostic traffic classification and anomaly detection work.

4.2.1 Social Bot Detection Approach.

Various approaches have been developed to identify social bots. According to the input data, they are often either content-based or structure-based [226], with a new trend on using crowdsourcing techniques as well. Table 16 shows a general comparison between these methodologies. BotFlowMon is the only approach that only relies on network traffic.

4.2.1.1 Content-Based Approach.

Content-based social bot detection approaches seek to detect behaviors, accounts, or account clusters associated with social bots using OSN content. Here, the OSN content refers to not only explicit information of an account, such as its profiles, URLs, and linguistic features of posts, but also implicit information of an account such as its clickstream, local graph structure, and user behavior. For example, research in [370, 431, 101] finds URLs in posts can help identify spam messages or social bot accounts; research in [358, 61, 371] leverages features related to account profiles or message content to determine if an account is actually a social bot; and research in [243, 429, 387] uses local graph structure information such as the number of followers and relationship between interactions for social bot detection.

Different methods have been used to model the differences of the OSN content between social bots and human users. While some used statistical analysis, including [372, 429, 371, 358], to be more flexible in tackling complicated features and more accurate, many leveraged machine learning techniques, such as random forest ([243, 101, 120]), SVM ([392, 61, 355]), and logistic regression ([370, 355]).

While often effective, content-based approaches have certain drawbacks. First, they require content data of OSN users. Compared with flow-level traffic data used in BotFlowMon, such data are usually privacy sensitive and could lead to potential privacy infringement. Second, they are vulnerable to adversarial attacks since social bot programs can mimic human users' behaviors to escape their detection.

4.2.1.2 Structure-Based Approach. The assumption of structure-based approaches is that social bot accounts (often referred as Sybil accounts in these approaches) can build connections between themselves arbitrarily, but it is difficult for them to establish or manipulate social relationship with human users [215]. The structural gap between social bot accounts and human accounts can then be used to identify social bots.

Structure-based approaches can be classified into two categories—random walk (RW) based and Markov Random Field (MRF) based. RW-based methods start walking from either a social bot node or a human user node and then use some classifiers to infer the labels of nodes along the path, as exemplified by research in [115, 85, 84]. MRF-based methods, such as those in [179, 388, 389], model the OSN structure as a Markov random field and use probability methods (e.g.Loopy Belief Propagation) to estimate each node's conditional probability of being a social bot. The two methods can be combined [390, 171].

Unlike content-based approaches, structure-based approaches are not vulnerable to content-oriented adversarial attacks. However, they have their own drawbacks: (1) Their aforementioned assumption is not always true. According to research in [231], it is easy for a social bot account to establish a relationship with a real user account. Moreover, once a real user account is compromised and becomes a social bot, it inherits all the social relationship of the real user. Consequently, structure-based approaches will have difficulty in detection social bots in such scenarios. BotFlowMon is not dependent on OSN structures, thus not subject to these issues. (2) They can only offer social bot detection at the account level, which is to determine whether an account is a real user or a social bot. BotFlowMon works at the behavior level which is a finer granularity. (3) They only detect social bot accounts with the change of social topology. If a social bot keeps posting messages without interacting with other accounts, they will fail to detect this social bot. In contrast, BotFlowMon detects social bots in real time soon after it receives NetFlow records of social bot traffic.

4.2.1.3 CrowdSourcing-Based Approach. A new trend in social bot detection is to utilize crowdsourcing [128, 394, 316], in which one can ask individual crowdworkers to judge whether a program is a bot or not and then aggregate the decisions from all crowdworkers. For example, research in [393] relies on a crowdsourcing layer to have individual users determine whether an OSN account is a bot account or not and a filtering layer to filter out unsatisfactory reports from individual users. Compared with BotFlowMon, a crowdsourcing approach tends to incur a high detection latency due to its interactions with human users, a high cost to pay crowdworkers, and privacy risks if distributing privacy-sensitive data to the crowd. Besides, a crowdsourcing approach usually detects social bots at account level, while BotFlowMon does so at behavior level.

4.2.2 Content-Agnostic Traffic Classification & Anomaly Detection. During the early development of traffic classification, content-agnostic traffic classifications was elementary and mainly focused on classifying traffic into a few frequently used applications, such as telnet, https, and BitTorrent. Different traffic data were used as input. For example, research in [113] used the size, inter-arrival time, and arrival order of IP packets; research in [53] leveraged packet header information; and research in [330] utilized metadata of packets, flows, and connections. Later, more approaches, such as those in [328, 217, 59], used increasingly popular NetFlow data to classify traffic. Compared with BotFlowMon which aims to classify traffic from the same application (traffic from social bots vs. traffic from human users), none of these approaches classify traffic from different groups of entities within the same application.

Like BotFlowMon, many traffic anomaly detection approaches are content-agnostic and use network flow information as input, such as those that detect distributed denial-of-service (DDoS) [70], botnet [144], worm [249], and cryptojacking [157]. However, the social bot anomaly that BotFlowMon tries to detect differs from these anomalies in fundamental ways, warranting a completely different detection approach. Foremost, any social bot anomaly is specific to an OSN application; to detect it using network flow records, one must extract OSN-specific behaviors from the records and detect behaviors that are anomalous . Existing content-agnostic traffic anomaly detection approaches do not study OSN-specific anomalies, making them unsuitable for detecting social bot traffic. Moreover, traffic caused by anomalies such as DDoS, botnet, worm, or crypto-mining is usually of different protocols, destinations, and underlying applications than those of legitimate traffic; social bot traffic, however, usually use the same protocols (e.g.HTTPS), destinations (e.g., Facebook), and underlying applications (OSN) as those of real OSN user traffic, making these attributes unusable for social bot detection. In fact, every content-agnostic traffic anomaly detection approach is designed for a specific type of anomaly and hardly interchangeable. For example, whereas content-agnostic botnet detection could use the trace of command and control channels (e.g.[369, 72]), IRC messages (e.g.[184]), or collective DNS queries

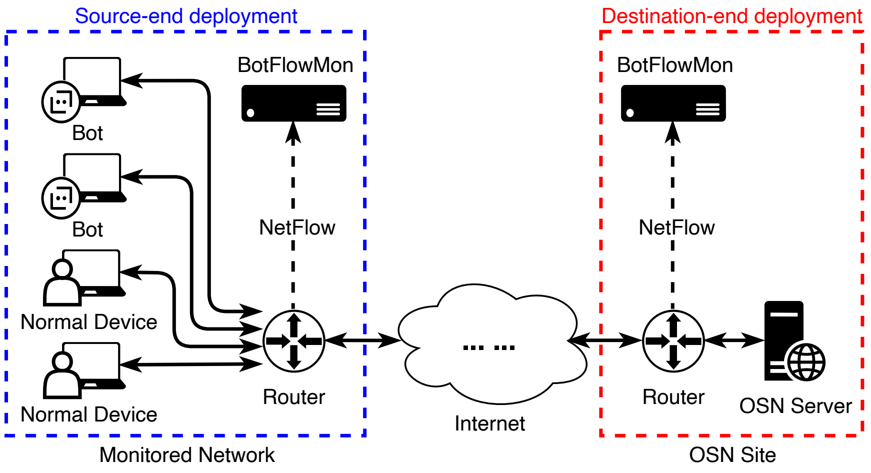


Figure 30. Operational models of BotFlowMon.

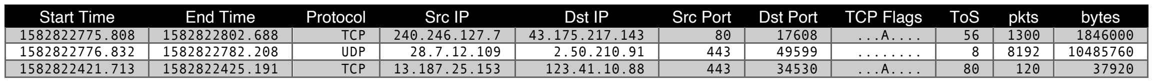
(e.g.[97]), none of these data or their properties applies to the detection of social bots.

4.3 BotFlowMon Design

4.3.1 Overview. In order to detect if any machine in a network is a social bot and producing social bot traffic, BotFlowMon can be deployed on any machine that can access and analyze the traffic flow data between the monitored network and OSN servers. Figure 30 shows two different operational models of BotFlowMon, one with BotFlowMon accessing traffic from the router of the monitored network at the source end (e.g.a campus network), the other from the router of an OSN server at the destination end (e.g.Facebook).

There are different traffic flow formats. We focus on the NetFlow [102] format, which is widely used for network traffic monitoring and analysis. Our design can easily extend to other flow formats such as sFlow [307].

Every NetFlow record logs information of a network flow inbound or outbound, including its start time, end time, number of packets, and number of bytes. It also records information from the TCP/UDP and IP headers of all the



Start Time	End Time	Protocol	Src IP	Dst IP	Src Port	Dst Port	TCP Flags	ToS	pkts	bytes
1582822775.808	1582822802.688	TCP	240.246.127.7	43.175.217.143	80	17608	...A....	56	1300	1846000
1582822776.832	1582822782.208	UDP	28.7.12.109	2.50.210.91	443	49599	....	8	8192	10485760
1582822421.713	1582822425.191	TCP	13.187.25.153	123.41.10.88	443	34530	...A....	80	120	37920

Figure 31. Fields of NetFlow records used by BotFlowMon.

packets in the flow, including source IP address, destination IP address, protocol, port numbers, type of service (ToS), and TCP flags. It will not access, read, or record the payload of any packet where the OSN content would be carried. As a result, a NetFlow record will *not* contain OSN content data such as OSN account profiles, Facebook or Twitter messages, or posted images. In another words, NetFlow records are content-agnostic. Using NetFlow records as input is thus privacy-preserving. Figure 31 shows the fields of NetFlow records that BotFlowMon uses, with a few examples.

Figure 32 illustrates the architecture of BotFlowMon. It encompasses two modes: **training mode** which uses labeled NetFlow data to derive a classification model and **detection mode** which uses the classification model to detect social bot flows from the input traffic flows. It also consists of *five* modules: preprocessing, flow aggregation, transaction fingerprint generation, transaction subdivision, and machine learning & classification. We detail each module below.

4.3.2 Preprocessing. ***Motivation.*** With the raw NetFlow records collected from a router as input, the preprocessing module needs to select records only related to OSN, group them by OSN users, and output them to the next module. As a router forwards packets and summarizes them into NetFlow records, there can be a vast amount of NetFlow records generated every second and these records can be noisy as they contain flows not toward OSNs or flows of irrelevant protocols and applications. The preprocessing module therefore must be both efficient and accurate in extracting OSN-related flows.

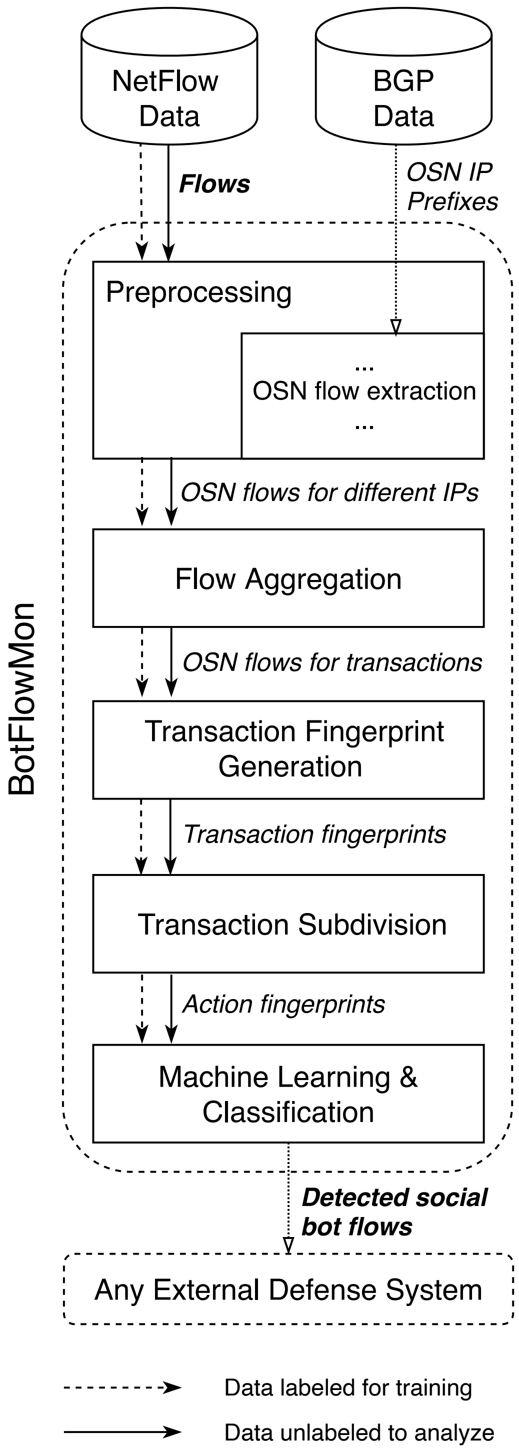

NetFlow Data
BGP Data
Flows
OSN IP Prefixes
BotFlowMon
Preprocessing
...
OSN flow extraction
...
OSN flows for different IPs
Flow Aggregation
OSN flows for transactions
Transaction Fingerprint Generation
Transaction fingerprints
Transaction Subdivision
Action fingerprints
Machine Learning & Classification
Detected social bot flows
Any External Defense System
Data labeled for training
Data unlabeled to analyze

Figure 32. BotFlowMon architecture. It has two modes of operation: training (data in dashed line) and monitoring (data in solid line).

Design. We preprocess the raw NetFlow data collected from a router as follows. We first extract the traffic flows only related to OSNs. After removing NetFlow records with zero bytes, zero duration, or irrelevant protocols (such as ICMP), BotFlowMon further discards NetFlow records whose source or destination IP address is not associated with the social network site(s) in question (e.g.Facebook or Twitter). One issue here is that each OSN site may own hundreds or even more IP prefixes and they may change over time. We leverage BGP stream [292] to obtain an OSN site's current IP prefixes and check if a NetFlow record's source or destination IP address matches one of the prefixes. In order to deal with a large number of flows efficiently, the matching process utilizes the longest prefix match algorithm [191], which is similar to the IP forwarding table lookup procedure when a router forwards packets based on their destination IP address. Our design allows BotFlowMon to preprocess NetFlow records in real time. We then group NetFlow records by OSN users, since these records may summarize traffic from thousands of users. Here, we define each OSN user by a unique combination of an IP address and a port number from the monitored network.

4.3.3 Flow Aggregation.

Motivation. Now that we have preprocessed NetFlow records to be composed of only those relevant to detecting social bot flows, we address the next challenge in that there is no sufficient information from data of *individual* NetFlow records to distinguish social bot flows from OSN flows generated by real users. As both social bot and real OSN user behaviors are conducted at the application level, their NetFlow records, which do not record application-specific data, can easily be indistinguishable.

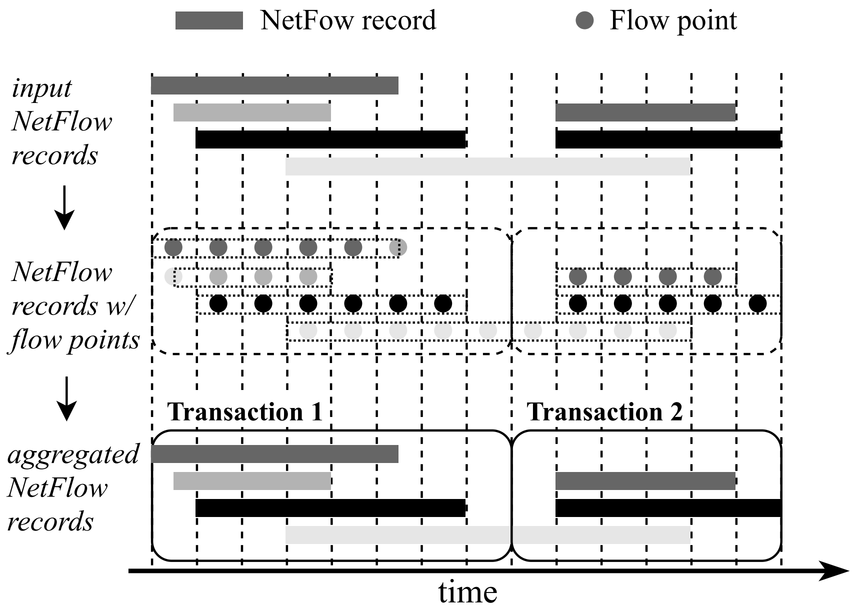


Figure 33. A flow aggregation example using modified DBSCAN. Every time bin is a small window (e.g.0.1 seconds). Six NetFlow records are clustered into two transactions by converting them to NetFlow records with flow points. (Each flow point has a different gray level, with a darker gray indicating a higher traffic volume.) One NetFlow record was fragmented into two records, each at a different transaction.

Design. We thus introduce the flow aggregation module in BotFlowMon to inspect *collective* OSN behaviors of flows in order to capture distinct patterns of social bot versus real user behaviors. It aggregates all the NetFlow records generated by the same **transactions**, so that we can inspect and compare transactions of a social bot against those of real users, including defining and comparing features of transactions. Here, a transaction is a sequence of actions by either a social bot or real user that are closely adjacent to each other. For example, it can be a user logging in her Facebook account and reading new posts on her Facebook wall, or a Twitter bot retweeting a spam link one hundred times within a short period.

In this module, we use modified DBSCAN [142] to aggregate/cluster flows into multiple transactions, with each transaction composed of multiple flows. Figure 33 shows an example. DBSCAN is a common density-based clustering algorithm that groups together adjacent high-density data points, where outliers are points that lie only in low-density regions. This procedure includes the following steps:

1. For every flow as described by a NetFlow record, we divide its duration into multiple time bins of equal length and define a flow point for each bin. Every flow point has a traffic volume derived from the bits per second (bps) of the flow multiplied by the bin's length.

2. We run the modified DBSCAN algorithm to group flow points into clusters, with each cluster composed of flow points that are closely adjacent to each other over a time window and have a high total traffic volume. In particular, for any time interval of ϵ seconds from the time window, the flow points that belong to the interval have a total traffic volume no less than $minPts$ bits.

3. Finally, we inspect the time window of every cluster from the above step. All the flows that fall within this window will belong to the same transaction.

4.3.4 Transaction Fingerprint Generation.

Motivation. With multiple transactions obtained from the flow aggregation module, every transaction may contain a different number of NetFlow records, further with its information in textual format (as every NetFlow record is textual). To make different transactions directly comparable to each other, BotFlowMon must define, extract, and normalize features from the aggregated flows of each transaction.

Table 17. 6×N matrix as a transaction fingerprint.

Features	Values			
1: outgoing bps	bps^o_{t1}	bps^o_{t2}	...	$bps^o_{t_N}$
2: outgoing pps	pps^o_{t1}	pps^o_{t2}	...	$pps^o_{t_N}$
3: outgoing ToS	tos^o_{t1}	tos^o_{t2}	...	$tos^o_{t_N}$
4: incoming bps	bps^i_{t1}	bps^i_{i2}	...	$bps^i_{t_N}$
5: incoming pps	pps^i_{t1}	pps^i_{i2}	...	$pps^i_{t_N}$
6: incoming ToS	tos^i_{t1}	tos^i_{i2}	...	$tos^i_{t_N}$

Design. BotFlowMon generates a **fingerprint** for each transaction. We design a **data fusion method** that derives an $f \times N$ matrix from every transaction and use this matrix as the fingerprint of the transaction. Here, N is the number of time bins of equal length within the time window of the transaction, which spans from the earliest start time to the latest end time among all flows in the transaction, and f is the number of features of the transaction over each time bin.

Table 17 shows a 6×N example transaction fingerprint matrix. Rows 1 to 3 are features of outgoing flows and rows 4 to 6 are features of incoming flows. Both use bits per second (bps), packets per second (pps) and type of service (ToS) as features. Note for any time bin there can be more than one flow active, thus the values of bps and pps (either incoming or outgoing) for that time bin should be respectively the sum of the bps and pps values of all flows active in that time bin. The outgoing or incoming ToS feature for a time bin, however, is not numerical, and its value is the ToS field (which is widely used for prioritizing traffic) and TCP flag of the flow that has the largest bps value during the time bin. However, because the usage of the ToS field has not been standardized, the ToS feature may not be reliable to help produce a transaction fingerprint. As such, we also introduce a 4×N matrix that does not include the ToS feature for incoming and outgoing flows.

Once a transaction fingerprint matrix is generated, it also must be normalized. In BotFlowMon, we use the quantile normalization approach to map *every* value in the matrix to a number between 0 and 255. Using the $6 \times N$ matrix from Table 17 as an example, we learned the distributions of bps, pps and ToS values using a 235-GB NetFlow dataset of campus traffic, obtained three functions $f_r(bps)$, $f_g(pps)$, and $f_b(tos)$ based on the distributions, and then normalized the bps, pps, and ToS values in the matrix to numbers between 0 and 255, respectively.

With all the values in a $f \times N$ transaction matrix between 0 and 255, we can easily visualize it for inspection and analysis. We can generate an image composed of two colorful bars in the standard RGB space, one for the incoming flows in the transaction and one for the outgoing flows, where each bar has a length of N pixels. In particular, we calculate the RGB value of pixel i (i=1,...,N) using all the values from the i-th column of the matrix. For example, for the $6 \times N$ matrix in Table 17, we can use the outgoing bps, pps, and ToS values in column i to derive the i-th pixel for the outgoing bar in the image as (R, G, B) = (255-bps, 255-pps, 255-ToS), which maps a bigger bps or pps value to a darker color.

Figure 34 is an example of visualizing a transaction fingerprint, which has 220 flows and represents a real user spending 35.74 seconds browsing Facebook. The beginning and ending positions of the image correspond to the starting and ending time of the transaction, respectively, with the upper bar about the outgoing traffic and the lower bar about the incoming traffic.

Figure 35 shows more examples of visualized transaction fingerprints from labeled ground truth. Transaction fingerprints in Figures 35a and 35b are two Twitter users messaging each other through Twitter Direct Messages, where Figure 35a is generated by a real user and Figure 35b is generated by a chatbot.

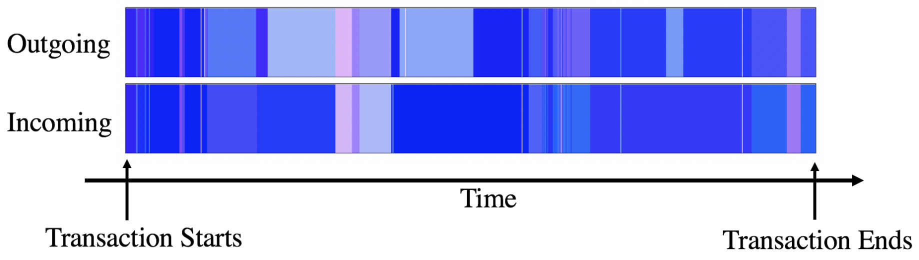


Figure 34. Visualizing the fingerprint of a transaction lasting 35.74 seconds with 220 flows.

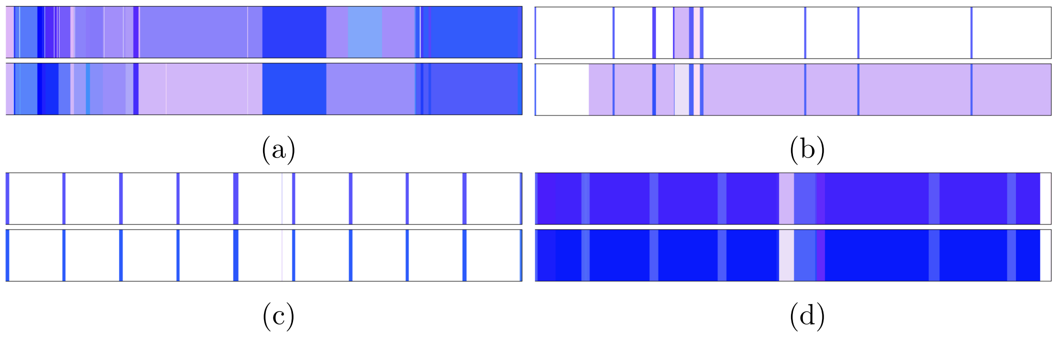


Figure 35. Transaction fingerprint image examples

Figure 35c is created by a social bot that uses APIs to tweet text messages every 3 seconds. Figure 35d is created by a bot that crawls the photo albums of friends and posts spam links at the same time. We can clearly see distinguishable patterns between the transaction fingerprints of a social bot and those of real users in terms of color depth, frequency, regularity, and density.

4.3.5 Transaction Subdivision. ***Motivation.*** Now that we have produced a normalized fingerprint matrix for every transaction, we observe there can be countless types of transactions since every transaction may contain an arbitrary number of actions of various types. In addition, each transaction can be of an arbitrary duration ranging from a few seconds to a few hours, which can lead to a huge amount of training data if to have enough training data for every possible duration range.

Design. We thus subdivide a transaction further into a sequence of primitive, short-lived behaviors called **actions**. Compared to countless types of transactions, there are only a limited number of types of actions, such as clicking a Like button, sending a tweet, or submitting a comment. As we will illustrate, as opposed to inspecting their transactions, it is much easier to differentiate social bots from real users through their actions in transactions. Once we tell social bot actions apart from real user actions, we can separate social bot transactions from real user transactions.

To subdivide a transaction into actions, we design a new clustering algorithm named **density-valley-based clustering** for this purpose. Compared with other density-based clustering algorithms such as DBSCAN [142] and OPTICS [50] that work by traversing density-connected areas, our algorithm clusters data points by finding the density valley between adjacent clusters. It does not require a density threshold parameter to conduct the clustering. Instead, it uses a valley point index ρ to identify the boundary of two clusters. Besides, our algorithm has a good performance in processing datasets whose density is not uniform.

This algorithm uses the following terms:

1. **Density of a data point:** The density of a data point p is the summation of the values of all data points within a radius of r from p. Using bps values as an example, $p.density = \sum_{dist(x,p)<r} x.bps$.

2. **Density of a cluster:** A cluster's density is the density of the point in the cluster that has the highest density.

3. **Potential point of a cluster:** Point p is a potential point of C if p is within radius r of a point $b \in C$.

4. **Valley point:** A data point p is a valley point of multiple clusters if p is a potential point of each of these clusters.

5. **Valley point competition:** When two clusters share a valley point, they "compete" to include the valley point as its member, with two possible outcome:

   (a) The two clusters merge into a new cluster, with the valley point now belonging to the new cluster;

   (b) The two clusters keep separate, with the valley point assigned to the cluster with less data points.

   Here, the two clusters keep separate if the density of the valley point is lower than a percentage ρ of, thus in sharp contrast to, the density of both clusters.

The algorithm works as follows, with its pseudocode in Algorithm 2. It takes a dataset D and a **radius threshold value** (r) as input. Using the $6 \times N$ or $4 \times N$ matrix from Section 4.3.4 as an example, D can be a set of data points where the i-th data point has a bps value that is the sum of the outgoing bps and incoming bps from the i-th column of the matrix. The algorithm then sorts all the data points in D, processes all the data points in the descending order of density, and forms and populates clusters with data points in D. If a data point is a valley point between two clusters, the algorithm then decides whether to merge the two clusters or still keep them using the valley point competition mechanism; if the two clusters do not merge, we also identify the subdivision moment where the two clusters meet.

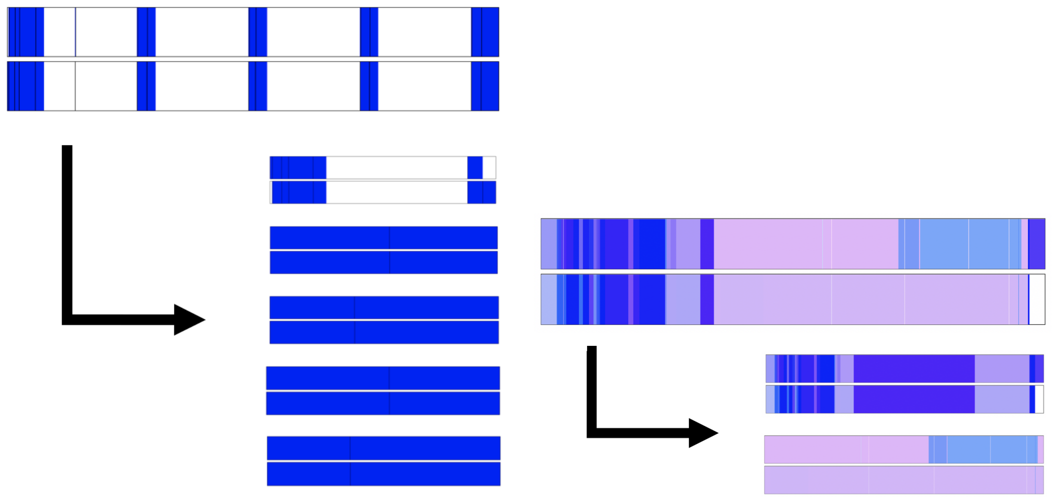
Algorithm 2 Density-valley-based clustering algorithm.

```
1: Input: dataset $D$, radius threshold value $r$, valley point index $\rho$
2: $C = \phi$                                  ▷ $C$ is a set to store clusters
3: Use $r$ to calculate the density of each data point in $D$
4: $D := Sort(D)$                ▷ Sort data in the decreasing order of density
5: for data point $e$ in $D$ do
6:     if $e$ is not a potential point of any cluster then
7:         Label $e$ as a member of a new cluster $c_e$
8:         Add cluster $c_e$ to $C$
9:     else if $e$ is a potential point of cluster $c_a$ then
10:        Label $e$ as a member of cluster $c_a$
11:    else if $e$ is a potential point of two clusters $\{c_i, c_j\}$ then ▷ Start the valley
   point competition
12:        if $e.density \leq \rho \cdot min(c_i.density, c_j.density)$ then
13:            Add $e$ to $c_i$ or $c_j$ that has less data points
14:        else
15:            $c_{new} = merge(c_i, c_j)$
16:            Add data point $e$ to cluster $c_{new}$
17:            Remove clusters $c_i$ and $c_j$ from $C$
18:            Add cluster $c_{new}$ to $C$
19:        end if
20:    end if
21: end for
22: return $C$
```

Finally, the algorithm outputs all the newly formed clusters. If D is a set of data points from a transaction's fingerprint, these clusters then represent actions of the transaction.

After subdividing a transaction into actions using the algorithm, we further ensure every action is short-lived according to the definition above. If an action produced from the algorithm is longer than 60 seconds, we further divide it into multiple actions at 60-second intervals.

Like a transaction represented by a transaction fingerprint matrix, an action can be represented by an **action fingerprint matrix**, which has the same look as a transaction fingerprint matrix as shown in Table 17.



(a) Social bot. (b) Real user.

Figure 36. Transaction subdivision examples.

Figure 36 shows two transaction subdivision examples. Figure 36a shows by subdividing the transaction fingerprint of a social bot (which is a post bot) into five actions, every action has a more outstanding pattern than the original transaction fingerprint. Figure 36b shows a transaction by a real user that is composed of two actions, where one was opening an OSN site and the other was scrolling down the page of the OSN site. We can see actions from real users present more complicated fingerprint images than those from social bots, making them easy to differentiate.

Algorithm 2 is dedicated to low dimensional datasets, such as the transaction fingerprint data. We further extend it in Appendix A to make it a general clustering algorithm that can deal with multi-dimensional datasets.

4.3.6 Machine Learning & Classification.

Motivation. With transactions subdivided into actions, the main challenge now becomes classifying transactions through their actions, while it is yet to be seen what models work best for classifying actions. Also, we need to devise an architecture when applying a model to process action fingerprints.

Design. We first classify actions into social bot actions and real user actions, and then classify the transactions based on how their actions are classified.

Action classification model. To classify actions we train an action classification model. The input to this model is a set of action fingerprint matrixes labeled as either social bot actions or real user actions. Since every action fingerprint matrix is nonlinear and high dimensional, we use Conventional Neural Network (CNN) [241] and Multilayer Perceptron (MLP) [205] as classification models. (BotFlowMon is not bound to MLP and CNN and could use other machine learning algorithms if applicable.)

CNN and MLP architecture. We use the Keras [16] library with TensorFlow [17] to implement CNN and MLP architectures. Figure 37 shows a CNN architecture that is composed of a convolution layer, a pooling layer and a flatten step to transform a fingerprint matrix into the input of a fully connected neural network, which consists of an input layer, multiple hidden layers (we evaluate the optimal number of hidden layers in Section 4.4.2.3), and an output layer to finally output the label for the fingerprint matrix. The MLP architecture is similar to the fully connected neural network in the CNN architecture, with the input layer being a flattened fingerprint matrix. For both architectures, the hidden layers use Leaky ReLU as its activation function so the model can converge quickly; the output layer uses the Sigmoid function to produce a probability that the fingerprint matrix is from a social bot. In addition, we train the CNN and MLP models with the stochastic gradient descent optimization algorithm and use the backpropagation algorithm to update the neural network.

Transaction Classification. With the action classification model trained, we then can classify if every action within a transaction is from a social bot or from a real user and decide whether the transaction is a bot transaction or a real user transaction. Apparently, if all actions are from a real user (or a social bot), we can safely determine the transaction is by a real user (or a social bot). However, a transaction may consist of actions of both types. We thus decide that a transaction is a bot transaction if more than a certain percentage of actions are from a social bot, and a real user transaction if otherwise. We define this percentage as the detection sensitivity rate γ of BotFlowMon.

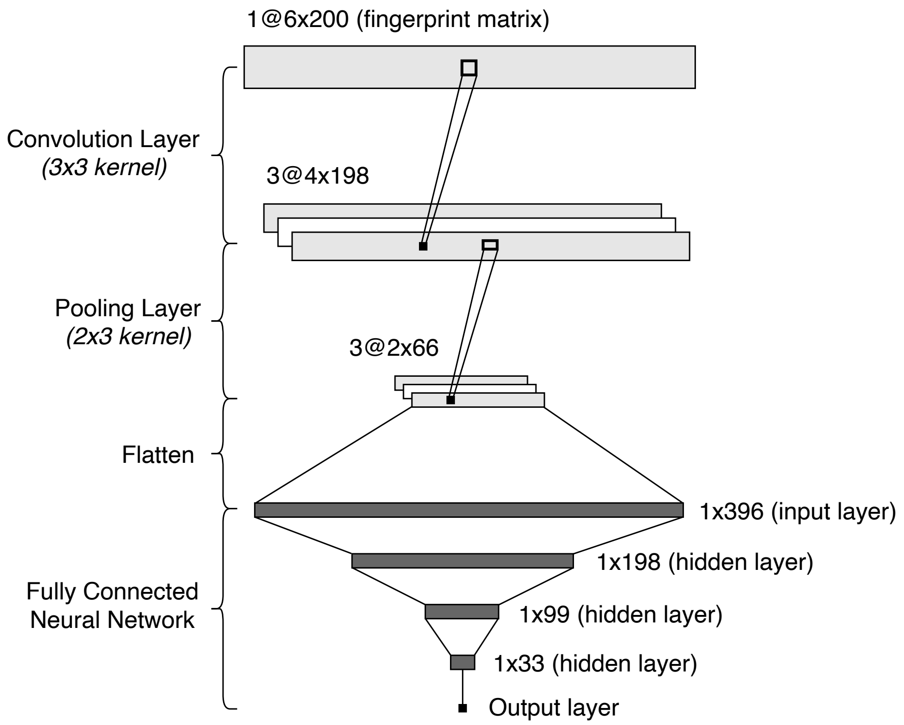


Figure 37. The CNN architecture of BotFlowMon (with example parameters).

4.4 Evaluation

We now evaluate BotFlowMon. We first introduce the dataset used in Section 4.4.1, then config and analyze parameters used in the BotFlowMon system in Section 4.4.2, present detection results and analysis in Sections 4.4.3–4.4.6, and finally discuss BotFlowMon's performance in Sections 4.4.7.

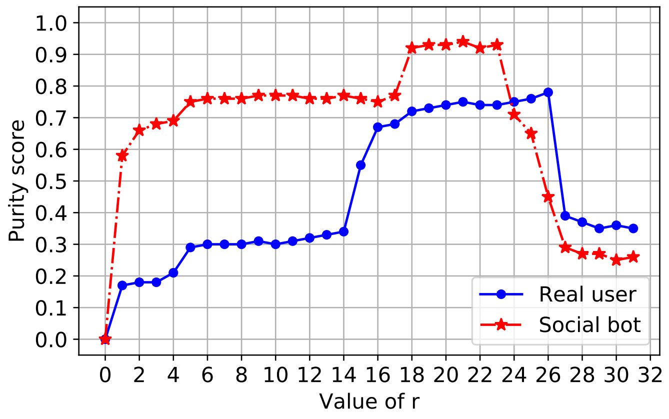


Figure 38. Purity scores with different r values.

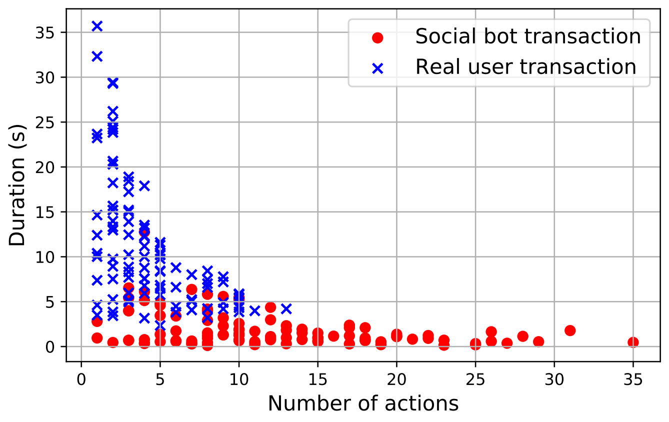


Figure 39. Scatter diagram for subdivision.

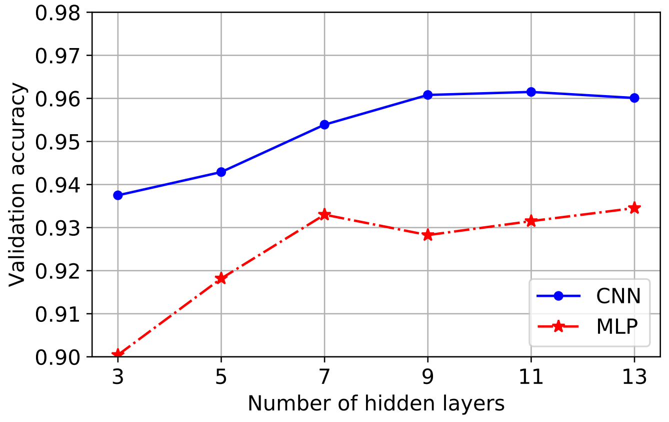


Figure 40. Validation accuracies with different numbers of hidden layers.

4.4.1 Data Source. The datasets we use to construct and test BotFlowMon come from two sources: (i) the traffic generated and gathered from our lab's computers and routers, which is a small experimental platform that has superior flexibility and convenience for simulation, data collection, and experiments; and (ii) datasets generated and collected from the campus network traffic of a large university, which offers data from realistic scenarios for analysis and verification.

We created and labeled the real user and social bot traffic flows as follows. For real user traffic flows, we recruited participants to manually conduct normal daily activities on Twitter and Facebook using our lab's computers. For social bot traffic flows, we deployed open-source social bot programs and homegrown bot scripts on the experimental platform and the campus network to conduct bot activities on Twitter and Facebook. We then collected and labeled the corresponding traffic as the ground truth. As described in Appendix B, we categorized social bots into five types according to their implementation mechanisms and simulated all of them. Besides, since the development and evaluation of BotFlowMon involved human subjects, we address the ethical and human subject issues of this process in Appendix C.

Table 18 shows the composition of the collected data. We collected 28 GB raw NetFlow data from our experimental platform and 507 GB raw NetFlow data from the campus traffic, both containing all traffic flows in the environment. After preprocessing, we had a dataset of 3.204 GB with 30,932,991 labeled NetFlow records for training and testing. The ratio of social-bot transactions to real-user transactions is approximately 7:3.

4.4.2 System Parameters Configuration and Analysis. The BotFlowMon system includes multiple system parameters for its different modules.

Table 18. Composition of collected data.

Raw NetFlow records			
Size of data	535 GB (campus network: 507 GB, lab platform: 28 GB)		
Labeled NetFlow records			
Size of data	3.204 GB (987.710 MB if removing irrelevant fields)		
# records	30,932,991		
Number of transactions (social bot : real user ≈ 7:3)			
Social bot	166,615	Real user	67,723

We first set up the empirical or default values for some system parameters in Section 4.4.2.1. We then investigate two key parameters more specifically: the radius threshold r in the transaction subdivision module in Section 4.4.2.2 and the number of hidden layers in CNN and MLP in the machine learning & classification module in Section 4.4.2.3.

4.4.2.1 System Parameters with Empirical or Default Values. Based on our empirical studies (of which we skip the details for space considerations), we set the following parameters as follows:

– For the flow aggregation module, we set the length of every time bin to be 0.1 seconds. This length provides a time granularity that is fine enough but not too small to skyrocket the computation cost of the system. This parameter is adjustable, as smaller time bins could lead to more precise results with a higher computational cost.

– For the DBSCAN algorithm in the flow aggregation module, we set ϵ to be 10-20 seconds. According to the user behavior models from the network perspective [336], users usually do not trigger NetFlow records in 10-20 seconds. This setup also helps us lean toward clustering the flows into transactions over a longer period rather than short ones, as the former is more friendly with the transaction subdivision module. Further, we set $minPts$ to be 1500 bits, which is a relatively small value for OSN traffic [336] in order to accommodate certain tiny streams between an OSN user and an OSN server.

– For the transaction fingerprint generation module, we set N as 200. Note a larger value of N will generate more accurate results but require more training data and computations.

We also set the default values of the following parameters:

– For the transaction subdivision module, we define ρ in valley competition mechanism to be 50% by default.

– For the machine learning & classification module, we set the detection sensitivity rate γ at 50% by default.

4.4.2.2 Subdivision Efficacy and Its Radius Threshold (r).

We evaluated the transaction subdivision module to see whether our density-valley-based clustering algorithm can divide the transaction fingerprints into action fingerprints correctly. Specifically, we studied how different values of radius threshold r in the algorithm could generate different subdivision results and potentially affect the detection outcome. Figure 38 shows the purity scores of the resulted clusters from subdivision with different r values. We use this formula to calculate the purity scores: $purity = \frac{1}{N}\sum_k max_j|c^k_{truth} \cap c^j_{algo}|$, where N is the number of input data points, c^k_{truth} is the k-th cluster from the ground truth, and c^j_{algo} is the j-th cluster generated by the algorithm. From Figure 38, we can see that the algorithm is indeed susceptible to the values of r and we can generate optimal results when r is in the range of 18 to 23. For transactions of social bots, the subdivision module works well when r is in this range and can achieve more than 0.90 purity of the resulting clusters. This is because social bots utilize APIs heavily, making their transaction fingerprints easy to subdivide. For real user transactions, however, the purity scores of the clusters derived from the algorithm are lower and more sensitive to the variation of r than those corresponding to bot transactions, with the maximal purity score to be 0.7832 when $r = 26$. One reason is that the boundaries of different actions in real user transactions are blurry in flow-level data. Almost all the OSN web sites preload content to real users dynamically, which can cause irrelevant NetFlow records to occupy the gap between actions and thus make the clustering in the algorithm less accurate.

However, the ultimate goal of subdivision is not partitioning all the transactions precisely. Instead, it is designed to make data more friendly to the machine learning process. We randomly sampled 100 real user transactions and 100 bot transactions, then recorded the number of actions and their average duration for each transaction after subdivision. Figure 39 is the scatter diagram of the result. We found that while the lengths of actions vary, actions from bots tend to have a short duration (0 to 15 seconds) and actions generated by real users usually have a long duration (0 to 40 seconds). Nonetheless, the durations of bot actions and real user actions are *comparable*, which makes it easy to compress actions of different durations into fingerprints of a fixed length *and* makes the subsequent machine learning module easy to converge.

4.4.2.3 Number of Hidden Layers in CNN and MLP. We investigated the optimal numbers of hidden layers of our CNN and MLP models as this parameter can significantly affect the accuracy of the models. In this evaluation, we took 80% of the labeled data as the training dataset, 10% of the data as the validation dataset, and the remaining 10% of the data as the testing dataset. For both CNN and MLP models, we started with three hidden layers. We trained each model with the training dataset and validated their accuracies with the validation dataset. We then kept adding new hidden layers to each model and repeated the procedure above until the validation accuracies converge.

Figure 40 shows the validation accuracies with different numbers of hidden layers. We can see the validation accuracy of CNN is already acceptable when the number of hidden layers is three and remains almost the same after it reaches nine. The MLP model's accuracy stabilizes when the number of hidden layers reaches seven. Therefore, we set CNN's default number of hidden layers at *nine* and MLP's default number of hidden layers at *seven*.

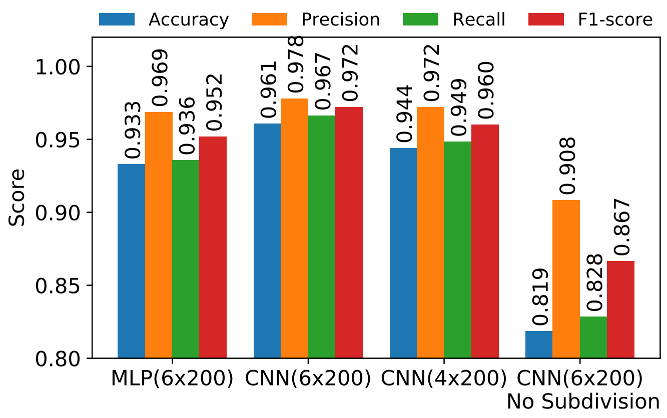


Figure 41. Detection scores for different detection models.

4.4.3 Classification Accuracy. We evaluated different classification models' abilities in detecting social bot transactions. Figure 41 shows the test results for machine learning & classification module with different classification models. The test dataset contains 10% of the labeled data, which has 16,661 social bot transactions and 6,772 real user transactions. Here, we use 6×200 transaction

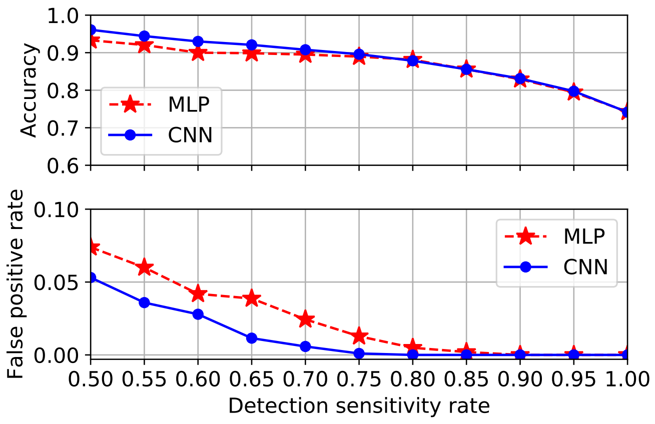


Figure 42. Detection accuracy and false positive rate with different detection sensitivity rates (γ).

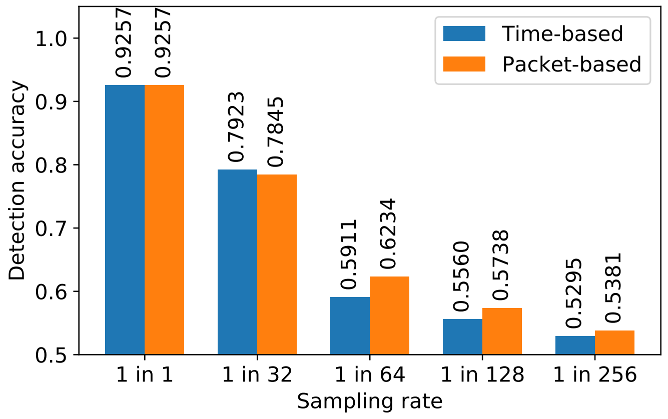


Figure 43. Detection accuracy with sampled NetFlow records (CNN with three hidden layers).

fingerprints which use incoming and outgoing bps, pps and ToS features. Both CNN and MLP models can correctly classify more than 93% of the traffic flows. However, CNN achieves a better significant result, with 96.1% of accuracy. One reason for this phenomenon is that CNN has convolutional and pooling layers to incorporate spatial information from action fingerprints, while MLP only takes flattened vectors as input, which disregards such spatial information. Moreover, we can clearly see that the subdivision procedure helps improve the accuracy for more than 10%. Table 19 shows other detailed evaluation scores of MLP and CNN for both 6×200 and 4×200 transaction fingerprints. We can see that CNN slightly surpasses MLP under all scoring criteria.

Table 19. True/false positive/negative rates for social bot traffic detection.

	MLP		CNN	
	6×200	4×200	6×200	4×200
True positive rate	0.9358	0.9318	0.9665	0.9485
True negative rate	0.9260	0.9221	0.9468	0.9334
False positive rate	0.0739	0.0778	0.0531	0.0665
False negative rate	0.0641	0.0681	0.0334	0.0514

Overall, BotFlowMon can detect social bots with high accuracy. With low false negative (or high true positive) rates as shown in Table 19, we can detect every single social bot transaction with a high probability. In particular, because a social bot usually creates multiple transaction fingerprints in a burst and we often only need to identify one of them to be able to trace the social bot, our chance of detecting the social bot is even higher.

A concern here is the false positive rates as we do not want BotFlowMon to mislabel real user traffic. An active social network user may generate hundreds of transactions per day, which means the false positive rate of the detection system needs to be low enough to avoid causing interruptions to real users. For this purpose, we can leverage the detection mechanism in the machine learning & classification module (Section 4.3.6); in particular, we can adjust the detection sensitivity rate γ to achieve zero false positive rate, which we further investigate in Section 4.4.4.1.

4.4.4 Putting BotFlowMon to Use.

We now study if putting BotFlowMon to use, what factors the user can config and adjust to suit their environment and need and how they may affect the detection of social bots.

4.4.4.1 Detection Sensitivity Rate (γ).

BotFlowMon allows its detection sensitivity rate γ to be adjusted to achieve different false positive rates. As stated in Section 4.3.6, given a sensitivity rate γ, BotFlowMon identifies a transaction as a social bot transaction if the percentage of social bot actions in the transaction is larger than γ. If we want to limit or eliminate false alarms, we can increase γ to sacrifice the detection accuracy a little bit for an extremely low (or even zero) false positive rate.

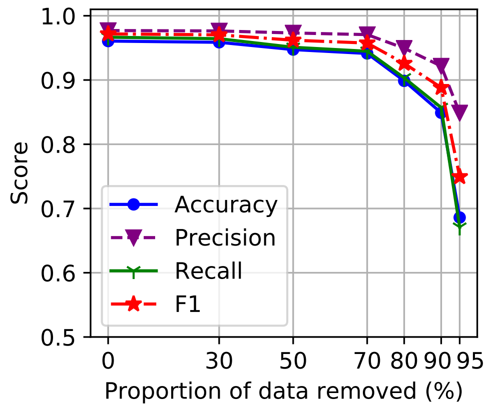
Figure 42 shows the accuracies and false positive rates of BotFlowMon with both MLP and CNN models when different γ values are used. We found from our experiments that the false positive rate of CNN will reach nearly zero when γ is no less than 0.75 (while the true positive rate is 0.91). Similarly, the false positive rate of MLP will reach zero when γ is no less than 0.85. If only when more than 75% of its actions are classified as bot actions will a transaction be labeled as a bot transaction, BotFlowMon with CNN will not generate any false alarm, while its accuracy will only drop down to 0.89. Therefore, a conservative approach to deploying BotFlowMon is to set the detection sensitivity rate γ at 0.75 if using

CNN (or 0.85 if using MLP), in which case the detection accuracy will be more than 0.89 if using CNN (or 0.85 with MLP) with no false positives.

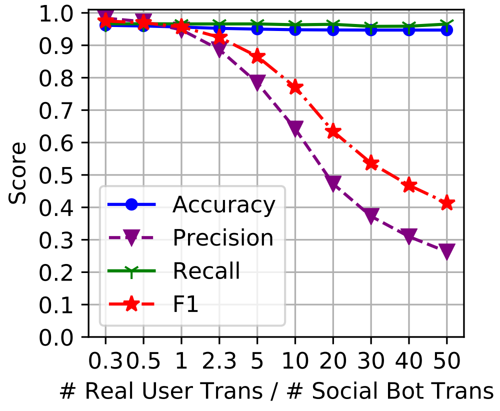
4.4.4.2 ToS Features. Due to reliability concerns with the ToS features (see Section 4.3.4), BotFlowMon may need to run without such features. We investigated BotFlowMon's detection efficacy without them. We thus used 4×200 transaction fingerprints which only included incoming and outgoing bps and pps features. Figure 41 shows the results, where the overall accuracy is 0.944, which is 1.7% lower than that with 6×200 transaction fingerprints. We can see without ToS features, we can have a classification model that is not only more usable but also still fairly accurate.

4.4.4.3 Detection with Sampled NetFlow Records. A reality factor in running BotFlowMon against NetFlow records as input is that many times due to computational and bandwidth pressure on routers, only sampled NetFlow records are available. We thus investigated BotFlowMon's accuracy with sampled NetFlow records.

We applied two sampling methods over the original NetFlow data from Table 18: time-based sampling that takes packets from a τ-second interval every $\tau{\cdot}x$ seconds, and packet-based sampling that takes one packet every x packets. We created multiple datasets using both methods with different sampling rates (i.e. $\frac{1}{x}$) and run BotFlowMon using the CNN classification model with three hidden layers. Figure 43 shows the results. We can see that as the sampling rate decreases the accuracies drop dramatically. BotFlowMon still works well with a low sampling rate at 1 in 32, but it becomes barely usable when the sampling rate becomes 1 in 64 or lower. On the other hand, as we pointed out in Section 4.4.3, because a social bot usually spawns multiple transaction fingerprints in a burst, among which only one needs to be identified to track down the social bot, we still have a reasonably high chance to include at least some bot transactions in the sampled input and be able to detect at least one of them.



(a) Detection scores with reduced training data.



(b) Detection scores with imbalanced test datasets.

4.4.4.4 Detection with Less Training Data. It is laborious to train a classification model with a large amount of labeled data. We investigated BotFlowMon's capability with less training data to understand the least amount of training data that would still lead to acceptable performance. Specifically, we removed different portions of data from the original training dataset and evaluated BotFlowMon's detection ability under each different training data size with default parameter values. Figure 44a shows the results. We can see BotFlowMon maintains a decent accuracy even after we removed 90% of the training data, but its accuracy degrades significantly when it is more than 90%. In another words, to maintain satisfactory performance the training needs at least 1500 transactions (250 MB) from real users and also this much from every type of social bot.

4.4.4.5 Detection with Imbalanced Datasets. When deploying BotFlowMon in different environments, the ratio of real user transactions to social bot transactions can change significantly. For instance, the proportion of social bot transactions in a VPS provider's network can be higher than that in residential networks. We therefore generated ten test datasets with different ratios to evaluate BotFlowMon's detection performance under such scenarios with default parameter values. Each of the generated test sets has 3000 samples. Figure 44b describes BotFlowMon's accuracy, precision, recall, and F1 scores with these test datasets. We can see that when the proportion of real user transactions increases, the accuracy and recall remain almost the same, but the precision and F1 score clearly drop. This phenomenon is expected as the false positive samples will dominate the detection errors when the number of false samples grows. To solve this problem, we can adjust the detection sensitivity rate discussed in Subsection 4.4.4.1 to reduce false alerts.

4.4.4.6 Selection of Parameters in New Environments. When transferring BotFlowMon to a different network environment, users do not need to reset the parameters if the following two conditions are met: (1) The input NetFlow records are of the same version and configuration. (2) Users are going to detect the same types of social bots on the same OSN sites. Otherwise, users need to reset the parameter values or retrain the detection model. To quickly set up the parameters in BotFlowMon and achieve acceptable performance in such scenarios, we suggest the users to begin with default parameters and use a conservative detection sensitivity rate γ (e.g.0.85 - 0.95). Such setting will sacrifice the true positive rate to decrease the false positive rate, enabling users to detect social bots to some extent while not being overwhelmed by false alarms. Later, after other parameters (e.g.the neural network) are fine-tuned according to the environment, users can adopt a radical detection sensitivity rate (e.g.0.5 - 0.75) for better accuracies.

4.4.5 Comparisons with Other Approaches. We have compared BotFlowMon with other social bot detection approaches, specifically content-based methods and structure-based methods (elaborated in Section 4.2). As illustrated in Figure 45, they each operate using different input at a different granularity in order to detect social bots.

We compared BotFlowMon with a content-based method described in [169], which we refer as SpamFilter. It utilizes four types of content features to detect social bots and achieves a true positive rate (TPR) of 80.8% and a false positive rate (FPR) of 0.32%. By analyzing their experimental data, an OSN account has to generate at least two transactions in order to create enough data to derive values of these features for SpamFilter to conduct its detection. We thus compare BotFlowMon with SpamFilter with two transactions.

As for structure-based methods, we selected SybilWalk [215] to compare, which achieves a TPR of 96% and a FPR of 1.3%. Here, in order to have enough data to derive values of features used by SybilWalk, an OSN account has to conduct at least four transactions, including account creation, profile initiation, and relationship establishment. We thus compare BotFlowMon with SybilWalk with four transactions.

Figures 46a and 46b illustrate the comparison results, where BotFlowMon uses the CNN model with 6×200 fingerprints and sets the detection sensitivity rate γ at 0.75. We can see the following. When there is only one transaction, only BotFlowMon can detect whether it is a social bot transaction with a TPR of 91% and a FPR of 0.1%. When there are two transactions, BotFlowMon can detect whether there is at least one social bot transaction with a TPR of 99.1% and a FPR of 0.2%, which is significantly better than SpamFilter's TPR (80.8%) and FPR (1.3%). And when there are four transactions, BotFlowMon can detect whether there is at least one social bot transaction with a TPR of 99.9% and a FPR of 0.5%, which is still significantly better than SpamFilter's TPR (96.3%) and FPR (2.6%) and is also better than SybilWalk's TPR (96.0%) and comparable to its FPR (0.3%). Overall, BotFlowMon outperforms both SpamFilter and SybilWalk with less transactions needed, a higher TPR, and a higher or comparable FPR.

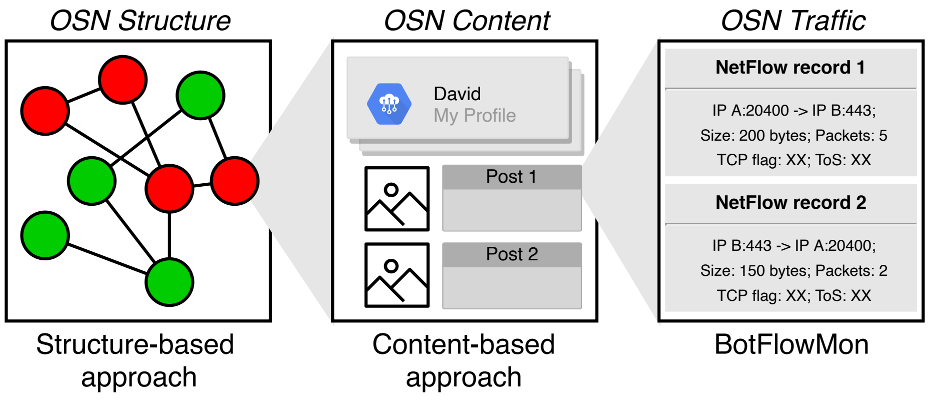
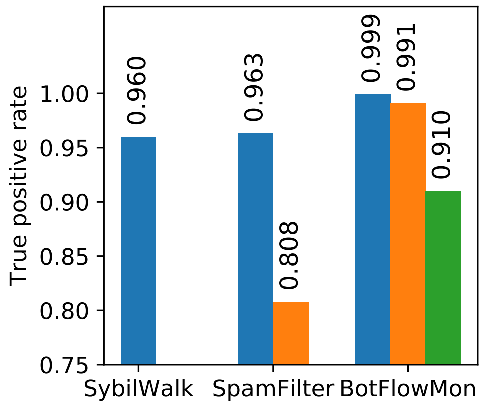
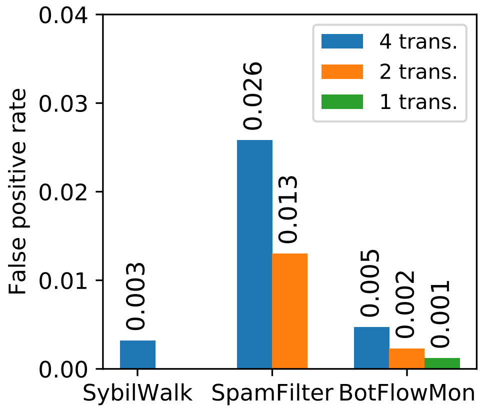


Figure 45. Input comparison of different social bot detection approaches.



(a) True positive rates.



(b) False positive rates

Figure 46. BotFlowMon vs. other approaches.

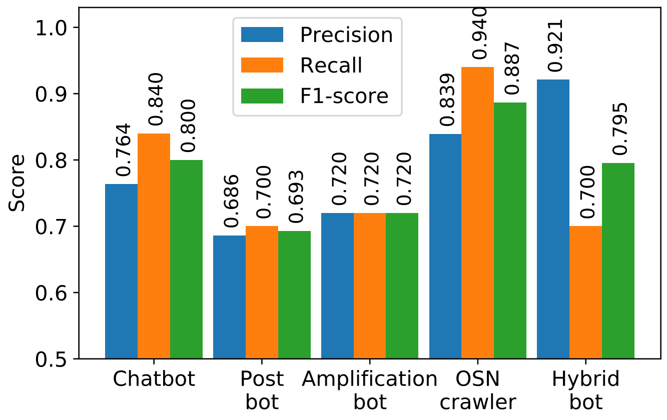


Figure 47. Multiclass classification scores for different categories of social bots.

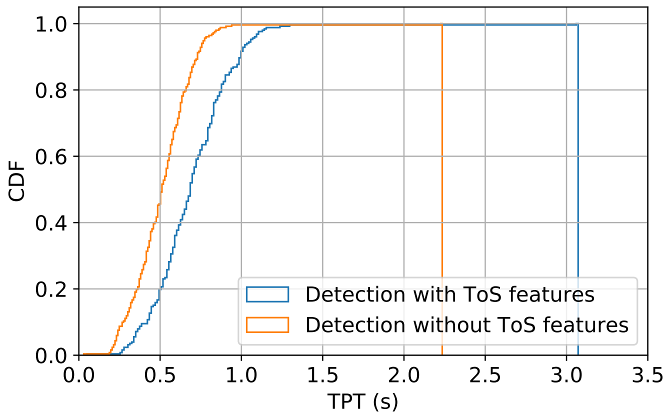


Figure 48. Cumulative distribution functions of TPT (time per transaction).

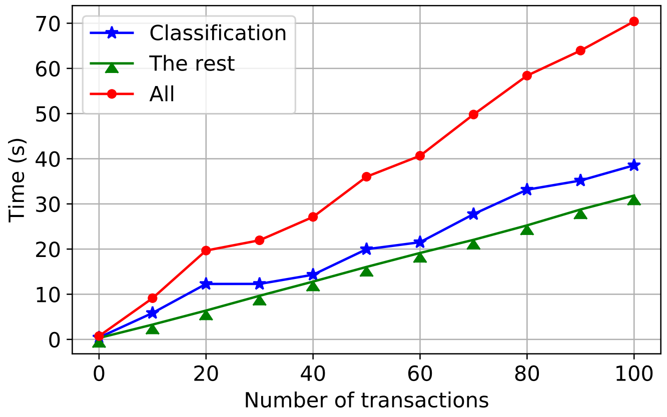


Figure 49. Performance of BotFlowMon and its modules (using CNN with ToS features).

4.4.6 Feasibility Study on Multiclass Classification of Social Bot Traffic.

Besides identifying social bot traffic flows, we investigated whether BotFlowMon may further classify the social bot traffic flows into different categories of social bots. By using the transaction fingerprints of different categories of social bots, we trained a multiclass CNN classification model that labels a social bot transaction fingerprint as one of the five following categories: chatbot, post bot, amplification bot, OSN crawler, and hybrid bot.

Figure 47 shows the multiclass classification scores for different categories of social bots. We used a test dataset that contains 250 bot transactions, with 50 transactions for each category. The overall classification accuracy is 78%. We can see from Figure 47 that this model achieves the best score for labeling OSN crawlers. This is because OSN crawlers usually have high frequencies, large packet sizes, and sharp contrast of outgoing and incoming traffic volumes, leading to the most differentiable traffic flows among the five types of social bots. The scores for chatbots and hybrid bots are less remarkable, but still decent. On the contrary, it is relatively hard to distinguish traffic flows from post bots and amplification bots; the API calls made by both types of bots, regardless of the functions invoked (e.g.liking a post, posting a message, following an account), will result in almost identical traffic flows.

In general, the accuracy for classifying categories of social bots is significantly lower than that of detecting social bots. These two tasks are fundamentally different. The former aims to classify the traffic as different types of social bot traffic, while the latter is focused on classifying the traffic as real-user traffic or social-bot traffic. Obviously, the difference between real-user traffic and social-bot traffic is more significant and easier to distinguish.

4.4.7 Time Complexity and Performance.

4.4.7.1 Time Complexity. The time complexity of BotFlowMon is $O(n)$, where n is the total number of input NetFlow records. The detailed analysis is in Appendix D.

4.4.7.2 Performance. We tested BotFlowMon's performance to see how fast it is in detecting social bot traffic. We tested it on a personal laptop with a quad-core 2.7-GHz CPU, 16-GB memory, but no GPU. We ran it in a single thread with the worst case for performance where *every* flow is an OSN flow.

We first evaluated how long it takes for BotFlowMon to process a single transaction, or **TPT (time per transaction)**. Note among the three factors we analyzed in Section 4.4.4, only the ToS features will affect TPT. We thus measured BotFlowMon's TPT with both 6×200 and 4×200 transaction fingerprints. Figure 48 shows the cumulative distribution functions (CDFs) of the TPT for both. Clearly, the TPT with 4x200 transaction fingerprints is shorter, where TPTs with 6×200 and 4×200 transaction fingerprints are 0.71 and 0.49 seconds on average, respectively.

Moreover, we measured the time spent on each module of BotFlowMon and found that about half time of a TPT is spent on the classification step. For the TPT with 6×200 transaction fingerprints, its 56.75% is from the final classification step and its 43.25% is from the rest operations.

Finally, Figure 49 shows that as the number of transactions increases, the total processing time increases basically linearly. Here, the processing time is the total time for the classification and the rest operations. The linear increase of the classification time is less stable than that of the rest operations, as the classification workload for each transaction depends on the number of its actions, which is uncertain.

Clearly, there is an ample space to improve both hardware and software support (e.g., using GPU and multithreading) for running BotFlowMon. Even without doing so, BotFlowMon is already fairly fast to detect social bot traffic in real time.

4.5 Discussions

In this section, we delve into the limitations and open issues associated with BotFlowMon, in addition to exploring its potential applicability across a diverse range of other applications.

4.5.1 Limitations and Open Issues. A primary contribution of BotFlowMon is to use data from network layers 3 and 4 to detect anomalies at network layer 7. However, BotFlowMon has some limitations:

1. As BotFlowMon is focused on distinguishing the social bot traffic from the real user traffic, its ability to identify different categories of social bots is not ideal (demonstrated in Section 4.4.6). To address this limitation, we could consider using social bot traffic identified by BotFlowMon to trace relevant OSN content and then leveraging the content to help classify social bots.

2. In fact, not all the social bots are necessarily malicious, and the boundary between "good" bots and "bad" bots can be blurry. To distinguish social bots with malicious intentions from innocent ones, not only will we very likely need content data of these social bots, but we also will have to apply techniques such as natural language processing to identify the intention behind a social bot.

3. If the training data of BotFlowMon does not include traffic of certain social bots, such as zero-day social bots, the system probably will not be able to detect them.

4. Adversaries could potentially circumvent BotFlowMon by programming their social bots to mimic the application-layer behaviors of real users, thus generating traffic patterns indistinguishable from real user traffic. In such scenarios, BotFlowMon might face challenges in accurately differentiating social bot traffic from that of genuine users. However, it's important to note that such instances are relatively uncommon in reality, primarily because replicating real user behavior is a complex task. Furthermore, real users' behaviors are typically inefficient concerning message dissemination or information crawling. Therefore, social bots that attempt to mirror these behaviors may struggle to achieve their intended goals with optimum efficiency.

BotFlowMon may also be enhanced with new functions and capabilities:

1. We may explore adding more features to BotFlowMon to improve its accuracy, such as the AS number of an IP, non-OSN traffic from the same IP, and time of the day.

2. We may add an online learning capability to BotFlowMon to make it adaptable to possible concept drifts.

3. We may study how to enhance BotFlowMon with adversarial machine learning techniques such that it is resilient against social bots that try to mimic real users.

4.5.2 Methodology Generalizations.

To identify social bot traffic from real user traffic, BotFlowMon proposes several new concepts, techniques, and traffic processing pipelines. These contributions can potentially be generalized to other FGTA applications or even other domains.

4.5.2.1 Traffic reassembly. Given that individual NetFlow records are insufficiently informative for detailed traffic differentiation, BotFlowMon employs a strategy that reassembles NetFlow records into transactions and further segments these transactions into actions for a more granular analysis. As detailed in Section 2.3.1, various existing FGTA systems implement this traffic reassembly process to obtain a more accurate representation of ongoing application or user behaviors. Consequently, our proposed traffic reassembly process could potentially be extended to additional FGTA applications such as website fingerprinting, application usage analysis, or device identification. This, in turn, would enhance the feasibility of utilizing flow-level traffic data in FGTA and improve classification accuracy. Furthermore, we've introduced a density-valley-based clustering algorithm to demarcate the boundaries of actions, as depicted in Algorithm 2 and 5. This clustering algorithm can be applied as a general-purpose tool for clustering any kind of data points and is particularly potent for handling datasets with uneven density distributions.

4.5.2.2 Fingerprint generation. Given the inherent complexity in directly processing traffic flow records using machine learning algorithms, BotFlowMon introduces an innovative fingerprint generation process. This process transforms a set of traffic flow records into a format more conducive to machine learning algorithms (e.g., CNN, MLP, etc.). The traffic fingerprint generated is not only easy to interpret but also simple to visualize, thereby aiding network administrators in comprehending the traffic patterns within their networks. This method of fingerprint generation is versatile enough to be adapted to other FGTA applications that utilize traffic flow records as inputs.

4.5.2.3 CNN for FGTA. BotFlowMon employs a CNN model to classify traffic fingerprints as either social bot traffic or authentic user traffic. While CNNs were initially conceived for image classification tasks, BotFlowMon evidences their potential applicability in the FGTA domain, thereby achieving respectable classification accuracy. Other FGTA applications may also stand to gain from the deployment of CNNs, particularly those that necessitate nuanced differentiation between various traffic classes.

4.6 Conclusions

As social bots become increasingly sophisticated and threatening, it's crucial to safeguard our online social ecosystems from potential disruptions or attacks. In response to this, we propose in this chapter a method named BotFlowMon for detecting social bots. This approach leverages FGTA and exploits the vast quantity of networking data to accurately identify the traffic attributable to OSN bots. As the networking data it processes are only information from network layers two and three and contain nothing about OSN content and structure, BotFlowMon departs from previous content-based, structure-based, and crowdsourcing-based solutions and is a content-agnostic, privacy-preserving, and efficient approach.

BotFlowMon devises several new techniques and algorithms, including an aggregation technique that derives transaction datasets from network flow records, a data fusion technique that extracts features from transactions datasets, and an innovative density-valley-based clustering algorithm that can both divide a transaction fingerprint into multiple actions and serve as a general clustering algorithm like DBSCAN to process other types of data. It can identify traffic flows from social bots with an accuracy of around 95%. It is also easy to deploy; any Internet service provider or enterprise networks, as long as they are able to access the traffic flow records such as NetFlow, can deploy BotFlowMon. Besides, it can monitor traffic of a network and detect social bots in real time; the study of the worst case on a testing machine shows BotFlowMon can determine whether a flow is from a social bot or not within 0.71 seconds on average after it sees the flow.

Additionally, this chapter broadens the application scope of FGTA, further demonstrating its versatility and effectiveness in the field of cybersecurity. Through these expanded use cases (Chapter III and IV), we showcase FGTA's adaptability to various cybersecurity challenges.

CHAPTER V
ON EXPLAINABLE AND ADAPTABLE DETECTION OF DISTRIBUTED DENIAL-OF-SERVICE TRAFFIC

Usability is another crucial aspect of FGTA. In this context, the usability of FGTA encompasses not only the accuracy, scalability, and efficiency of the FGTA system, but also its explainability and adaptability. Explainability refers to the system's ability to elucidate its decisions or results to users, enabling them to comprehend the system's operations, verify its correctness, and, if necessary, intervene. Adaptability, on the other hand, speaks to the system's capability to adjust to new environments or scenarios without necessitating system retraining, thereby facilitating users to readily deploy the system in novel environments or situations. Regrettably, these vital elements are often neglected or insufficiently addressed in existing FGTA literature and systems.

In this chapter, we demonstrate, through a practical case study, how to augment the explainability and adaptability of FGTA, making it more practical and applicable in real-world scenarios. Specifically, we introduce a novel FGTA approach for classifying DDoS traffic, which is characterized by its superior explainability and adaptability. The proposed detection approach outputs explanatory information that enables network administrators to easily inspect detection results and make necessary interventions. Moreover, this approach is adaptable in that users do not need to retrain the detection model to have it fit with a new network environment.

This chapter is derived in part from the following published and unpublished articles:

– Unpublished as Yebo Feng, Jun Li, Devkishen Sisodia, and Peter Reiher. “On Explainable and Adaptable Detection of Distributed Denial-of-Service Traffic.” *IEEE Transactions on Dependable and Secure Computing*, 2023. **In submission**.

– Published as Yebo Feng, and Jun Li. “Toward explainable and adaptable detection and classification of distributed denial-of-service attacks.” *In Deployable Machine Learning for Security Defense: First International Workshop (MLHat 2020), Proceedings 1, pp. 105-121*, 2020 [149].

– Published as Yebo Feng, and Jun Li. “Towards explicable and adaptive DDoS traffic classification.” *In The 21st Passive and Active Measurement Conference as poster*, 2020 [150].

I am the leading author of the above articles. Most content of this chapter was written by me, and I was responsible for conducting all the system designs, implementations, evaluations, and the presented analyses.

5.1 Introduction

Distributed denial-of-service (DDoS) attacks pose a severe security problem on today’s Internet and can render servers, network infrastructure, and applications unavailable to their users. They overwhelm the targeted machine or network resources with excessive traffic, thereby preventing legitimate traffic from being processed [14]. Cisco indicated in their March 2020 white paper [29] that the frequency of DDoS attacks had increased more than 2.5 times and the average size of DDoS attacks had approached 1 Tbps over the last three years.

Of foremost importance in DDoS defense tasks are to detect DDoS, classify DDoS sources, and do so accurately and quickly. Decades of research and industry efforts have led to a myriad of DDoS detection and classification approaches. In recent years, many researchers have begun to harness machine learning algorithms, such as support vector machine (SVM), Naive Bayes, convolutional neural network (CNN), etc., on big data in detecting and classifying DDoS attacks (e.g.[361, 435, 131]). The evaluations of such approaches demonstrate their strong ability in extracting useful knowledge from massive training data and decent recall scores in detecting a variety of DDoS attacks.

Unfortunately, the negative aspects of most learning-based approaches are also apparent. Firstly, many learning-based approaches may not be well-suited for practical applications, as their detection results are often difficult to interpret, resembling black boxes [138, 100]. As a result, extracting explanatory information from the detection outputs generated by these methods (e.g., deep neural networks and deep recurrent neural networks) is challenging. In real-world deployments, network administrators particularly need good explainability, as they usually have to manually review and verify DDoS detection results, including eliminating false alarms and avoiding severe collateral damage due to filtering traffic from legitimate users. This is especially true for large-scale networks, such as Internet service providers (ISPs) and Internet exchange points (IXPs), where a single filtering rule can disconnect a considerable number of IP addresses, making network administrators hesitant about which actions to take. According to previous literature [132, 334, 374] and our analysis (Section 5.3), detection approaches with useful explanatory information should possess three features to help network administrators make appropriate and timely decisions:

- **Transparency**: The detection model should allow users to gain clear insight into the traffic processing procedures, intuitively illustrating all network contexts and situations.

- **Traceability**: The detection outputs should help users quickly understand the detection logic and indicate root causes.

- **Heuristic**: The detection outputs should help users make applicable decisions to address the ongoing attack by quantifying the attack status, attack intensity, and the mitigation cost-effectiveness.

However, most existing DDoS detection approaches struggle to meet these requirements.

Secondly, most learning-based approaches lack adaptability. Their performance is heavily dependent on the coverage and applicability of the training data. Considering that DDoS attacks are diverse and network traffic regarded as DDoS in one environment might be considered legitimate in another (and vice versa), few of the current learning-based approaches can effectively adapt a DDoS detection model trained in one environment to fit in a new network environment. This limitation leads to poor detection accuracy or the need for lengthy retraining.

To address these missing gaps, we design a machine-learning-based DDoS detection and classification approach that is not only effective, but also explainable and adaptable. Specifically,

1. With network traffic flows summarized into traffic profiles, our approach can detect an DDoS attack (i.e.detection) and identify DDoS sources (i.e.classification) accurately and quickly. To detect DDoS, it enhances the k-nearest neighbors (KNN) algorithm to place traffic profiles into different

regions into the searching space and can categorize traffic profiles to be benign or malicious and detect if the current traffic profile corresponds to a DDoS attack. Furthermore, to improve the efficiency of the detection process, it introduces a k-dimensional (KD) tree algorithm to convert the KNN detection model into a semi-decision tree, which significantly reduce the time complexity of traffic monitoring to $O(d)$ in most cases, where d is the depth of the semi-decision tree. If a DDoS attack is detected, to identify DDoS sources, it will sort the traffic sources (i.e.senders' IP addresses) based on risk levels to minimize collateral damage, and iteratively identify and remove the malicious IP addresses until the traffic profile returns to a benign position in the KNN searching space.

2. Our approach is highly explainable, characterized by its *transparency*, *traceability*, and *heuristic qualities*. These attributes enable the generation of intuitive explanatory information, allowing network administrators to easily understand and act upon them. Upon detecting a DDoS attack, our approach not only sends an alert message but also provides a *risk profile*, a *visual detection model*, and a *status graph* to elucidate the attack. The risk profile represents the shortest Euclidean distance from the current traffic profile to a benign region in the KNN search space, assisting network administrators in quantifying the attack's magnitude and the associated mitigation costs. The visual detection model clarifies the detection logic, network context, and root causes by illustrating the relative distances from the current traffic profile to illegitimate and legitimate groups. Generated using principal component analysis (PCA) projection, the status graph concisely and intuitively depicts the attack stage, intensity, and cost-effectiveness of mitigation efforts.

3. Our approach is adaptable in that the detection and classification model derived in one environment can port to another environment without re-training. It allows direct modifications on the KNN searching space and enables users to leverage a variety of prior knowledge to evolve the detection model.

We evaluated our approach in both simulated environments and the real world. We first trained and evaluated our detection model with representative DDoS datasets in simulation environments. The results indicate that the detection model can achieve an accuracy of 0.956 and a recall score of 0.920 even when detecting some application-layer DDoS attacks. We then conducted a human subject study with questionnaire surveys to evaluate its explainability. The results demonstrate that the explanatory outputs can effectively help users understand not only the intensity, stage, and confidence level of the attack, but also can help them make suitable mitigation strategies quickly. Furthermore, as this approach is easily adaptable to a new environment, we transferred the model (with merely some measurement data as input) to a real-world network environment at the Front Range GigaPoP (FRGP) [5], a regional IXP in USA. We successfully detected most of the real-world DDoS attacks from February 24 to May 21, 2020, which we verified with the IXP. The latency of detection is also low—e.g.with a throughput of 100 Gbps, our approach can complete detection in around five seconds.

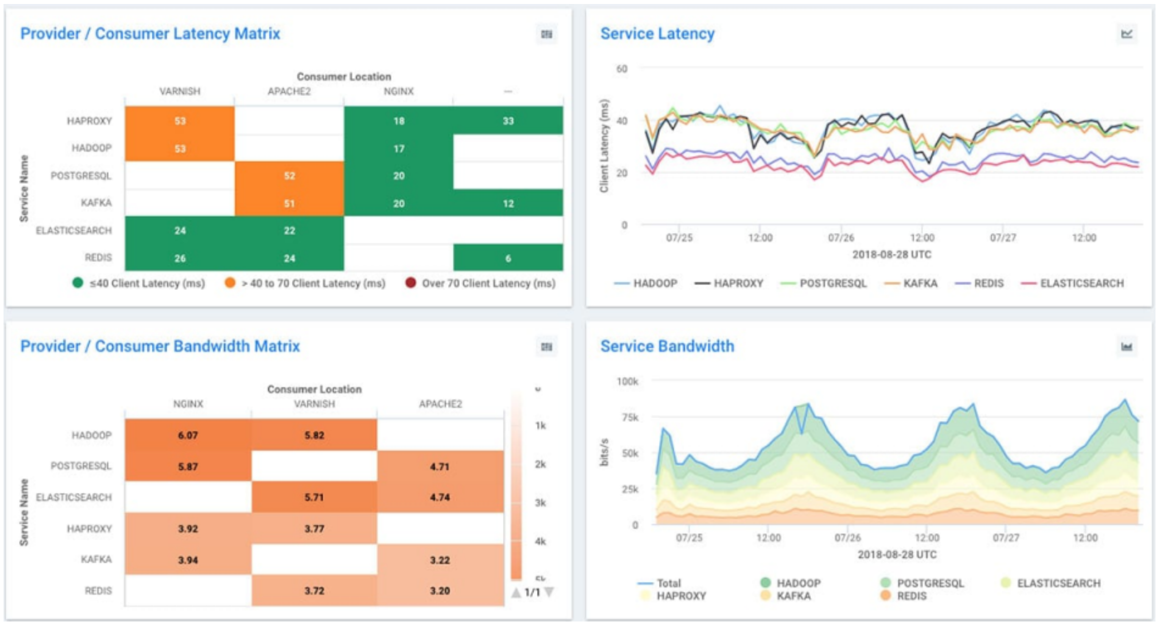
The structure of this paper is organized as follows: We commence by introducing relevant work in Section 5.2, followed by the presentation of the threat and defense model in Section 5.3. Subsequently, we elucidate our method design in Section 5.4. Evaluation of our approach is discussed in Section 5.5. We then turn our attention to the limitations and potential broad applicability of our approach in Section 5.6, and we conclude the paper in Section 5.7.

5.2 Related Work

Using network traffic data to detect DDoS attacks is a technique that is widely used in the security community. From the perspective of operating principles, we can further classify the existing DDoS detection approaches into *statistical approaches*, *rule-based approaches*, *learning-based approaches*, and *soft-computing-based approaches*. We discuss the advantages and disadvantages of each approach in detail.

5.2.1 Statistical Approaches. Statistical approaches detect DDoS attacks by exploiting statistical properties of benign or malicious network traffic. These approaches are straightforward and dominated the early development of DDoS detection. Generally, these approaches build a statistical model of normal or malicious traffic and then apply a statistical inference test to determine if a new instance follows the model [70]. For example, D-WARD [273] uses a predefined statistical model for legitimate traffic to detect anomalies in the bidirectional traffic statistics for each destination with periodic deviation analysis. Chen [91] proposed a DDoS detection method based on the two-sample t-test, which indicates that the SYN arrival rate of legitimate traffic follows the normal distribution and identifies a DDoS attack by testing the distribution compliance. Zhang *et al.* [438] proposed a detection method by applying the Auto Regressive Integrated Moving Average model on the available service rate of a protected server.

Statistical approaches can provide interpretable results by outputting abundant metrics to describe the current network situation, such as shown in Figure 50. These metrics primarily function as network measurements, assisting



Provider / Consumer Latency Matrix (Consumer Location × Service Name)

Service Name	VARNISH	APACHE2	NGINX	—
HAPROXY	53		18	33
HADOOP	53		17	
POSTGRESQL		52	20	
KAFKA		51	20	12
ELASTICSEARCH	24	22		
REDIS	26	24		6

● ≤40 Client Latency (ms) ● > 40 to 70 Client Latency (ms) ● Over 70 Client Latency (ms)

Provider / Consumer Bandwidth Matrix (Consumer Location × Service Name)

Service Name	NGINX	VARNISH	APACHE2
HADOOP	6.07	5.82	
POSTGRESQL	5.87		4.71
ELASTICSEARCH		5.71	4.74
HAPROXY	3.92	3.77	
KAFKA	3.94		3.22
REDIS		3.72	3.20

Figure 50. Partial outputs of Kentik [6], a popular traffic monitoring tool that can detect and mitigate DDoS attacks.

network administrators in grasping the network context. However, they are often not arranged in a concise and heuristic manner that would enable the identification of root causes and cost-effective mitigation strategies. As a result, skilled analysts are still indispensable for extracting valuable insights from these metrics to not only understand the importance of the alarms but also determine the appropriate course of action. Another limitation of statistical approaches is that as DDoS attacks evolve, traffic generated by sophisticated DDoS attacks may not always exhibit significant statistical deviations across various aspects. Consequently, traditional statistical DDoS detection methods might struggle to accurately identify modern DDoS attacks.

5.2.2 Rule-based Approaches. Rule-based approaches formulate noticeable characteristics of known DDoS attacks and detect actual occurrences of such attacks based on those formulated characteristics. For example, NetBouncer [373] detects illegitimate clients by conducting a set of legitimacy tests on the clients; If a client fails to pass these tests, it will be considered malicious until a particular legitimacy window expires. Wang *et al.* [396] detect DDoS with an augmented attack tree (AAT), which captures incidents triggered by DDoS traffic and the corresponding state transitions from the view of network traffic transmissions. Limwiwatkul *et al.* [253] detect ICMP, TCP and UDP flooding attacks by analyzing packet headers with well-defined rules and conditions. However, due to the growing diversity of DDoS attacks, rule-based approaches face challenges in summarizing and formulating the features of all possible attack types. Consequently, they are being gradually replaced by learning-based or soft-computing-based methods.

5.2.3 Learning-based Approaches. Over the past few years, more and more researchers have begun to leverage machine learning techniques to model, mitigate, and detect DDoS attacks (e.g., [131, 195, 436, 260, 154, 338, 230, 55, 270, 412]). Some of these methods (e.g., [244, 446, 68]) utilize unsupervised learning algorithms to distinguish anomalies from normal traffic, as such algorithms do not require training before the detection. However, unsupervised-learning-based approaches are sensitive to the selected features and the background traffic. On the other hand, supervised-learning-based approaches may struggle to provide users with explainable detection results, as the prevalent machine learning algorithms (e.g., linear regression [276], multilayer perceptron [172], convolutional neural network [232], graph convolutional network [86], etc.) often resemble black boxes in their functionality. In real-world deployments, explainable results are critical for attack mitigation, because network administrators usually need to manually review the detection results in order to eliminate false positives and maintain the usability of their network infrastructure.

Recently, there has been a surge of efforts aimed at enhancing the explainability of machine learning algorithms. For example, Nguyen et al. [286] proposed a machine learning-based anomaly detection approach capable of informing users about the types of detected anomalies and the significant features contributing to the detection process. Ribeiro et al. [322] introduced Local Interpretable Model-agnostic Explanations (LIME), which offers insights into machine learning model predictions by generating locally interpretable explanations, enabling users to better comprehend the decision-making process of complex models. Additionally, Lundberg et al. [262] presented SHapley Additive exPlanations (SHAP), a unified method for explaining the output of various machine learning models. Nevertheless, some of these approaches have not been implemented for DDoS detection, some of their explanations may not be suitable for DDoS detection scenarios, or some may not completely fulfill the transparency, traceability, or heuristic requirements.

In addition, the applicability of these machine learning algorithms highly depend on the training data and training environment, which means it is difficult to quickly transfer a detection model trained in one network environment to another network environment.

Therefore, although most learning-based approaches are usually accurate in detecting DDoS attacks, they are not easily deployable in real-world environments. As opposed to these previous learning-based approaches, our approach focuses on the explainability and adaptability of the detection model.

5.2.4 Soft-computing-based Approaches.

Soft computing is a term for describing the use of approximate calculations to provide imprecise but usable solutions to complicated computational problems. Such approaches match the general goal of DDoS detection, which is to identify attack sources while allowing only a few false positives and false negatives. Soft computing approaches can be an ensemble of statistical, rule-based, and learning approaches. For example, Jalili *et al.* [213] use statistical preprocessing to extract features from the traffic, and then utilize an unsupervised neural network to classify traffic patterns as either malicious or legitimate. Kumar *et al.* [233] utilize a resilient back propagation neural network as the base classifier, then propose RBPBoost to combine the outputs, and Neyman Pearson cost minimization strategy to generate the final classification decision. Shiaeles *et al.* [344] detect DDoS attacks based on a fuzzy estimator using mean packet inter-arrival times within 3-second detection windows. Just like learning-based approaches, soft-computing-based approaches also have the disadvantage of poor explainability, making them difficult to deploy in real-world scenarios.

5.3 Threat and defense models

In this section, we begin by presenting the threat models associated with DDoS attacks, followed by a description of the defense model of the proposed approach.

5.3.1 Threat model.

DDoS attacks are malicious efforts aimed at disrupting the normal operation of a targeted server, service, or network by inundating it with an overwhelming volume of internet traffic. These attacks are carried out by multiple systems, often compromised by malware and controlled by a single attacker known as a botmaster. The compromised systems, referred to as bots, constitute a network called a botnet. The primary objective of a DDoS attack is to render the target's resources inaccessible to legitimate users, resulting in downtime and potential financial or reputational harm.

Regarding attack methodologies, DDoS attacks can be categorized into three main types:

- **Volumetric attacks** strive to overwhelm the target's bandwidth by generating an immense volume of traffic, impeding legitimate users from accessing the targeted service. Examples of volumetric attacks include UDP floods and ICMP floods [441].
- **Protocol attacks** leverage vulnerabilities in network protocols to consume resources or cause network disruptions. Examples of such attacks include SYN floods, which target the TCP handshake process, and Ping of Death attacks that transmit oversized ICMP packets [329].
- **Application-layer attacks** focus on specific applications or services, overloading them with seemingly legitimate requests. These attacks demand fewer resources for execution but may pose greater challenges in detection and mitigation. Examples include HTTP GET floods, Slowloris attacks, and DNS query floods [310].

DDoS attacks can cause significant harm to victims, leading to service disruptions, revenue loss, reputational damage, and increased security expenses. Therefore, it is crucial for organizations to implement strong security measures to minimize the impact of such attacks.

5.3.2 Defense model.

The defense model of the proposed approach operates as follows:

1. Initially, the network administrator deploys the proposed approach on the network to be secured. It is important to note that this network may differ from the one where the approach was trained.

2. The approach continuously monitors network traffic, identifying any DDoS attacks aimed at targets within the protected network.

3. Upon detecting a DDoS attack, the approach classifies the DDoS traffic and sends the classification results as mitigation rules to upstream routers or Internet service providers.

4. The mitigation rules typically include malicious IP addresses/IP prefixes or malicious traffic flows. These rules are then applied to network traffic to alleviate the DDoS attack, preventing it from reaching its intended victim.

5.3.2.1 Adaptability to the network to be secured. The network requiring protection might not be the same as the one on which the approach was trained. This can occur when the approach is trained using a public dataset that may not accurately represent the specific network to be secured. Consequently, it is essential for the approach to rapidly adapt to the network in need of protection without necessitating extensive time, a large volume of training data, or numerous fine-tuning processes.

5.3.2.2 Minimizing Collateral Damage and Verifying Results. In the context of DDoS attacks, collateral damage refers to the unintended consequences of mitigation rules on legitimate traffic. If a rule inadvertently blocks a valid IP address, it can disrupt genuine traffic, potentially causing more harm than allowing malicious traffic to reach the intended target.

To minimize collateral damage, network administrators typically need to manually verify detection results before implementing them as mitigation rules (at steps 2 or 3). Several factors should be considered during the verification process, including:

– **Network Context:** Administrators should evaluate the network context, taking into account factors such as attack intensity, the number of attack sources, current network throughput, and more. This information is vital for understanding the attack's impact and the necessity of mitigation, allowing for appropriate planning and next steps.

– **Detection Logic:** Understanding the detection logic of the chosen approach is essential for administrators to determine the reliability of the results. Additionally, this information can help identify the root cause of the attack, aiding in the elimination of potential false positives. For example, during high-traffic periods, duplicate user requests may be misclassified as application-layer DDoS attacks (i.e., flash crowds). This type of misclassification can be avoided by quick verification.

– **Mitigation Cost-Effectiveness:** Since mitigation rules can lead to collateral damage and additional costs, administrators should weigh the cost-effectiveness of the proposed rules. In some cases, even when the network is under attack, the system may have enough redundancy to cope during periods of low activity. In such instances, administrators may opt not to apply mitigation rules to avoid unnecessary collateral damage.

Thus, the proposed approach should offer adequate explanations to aid administrators in verifying the results. Specifically, it needs to be *transparent* for

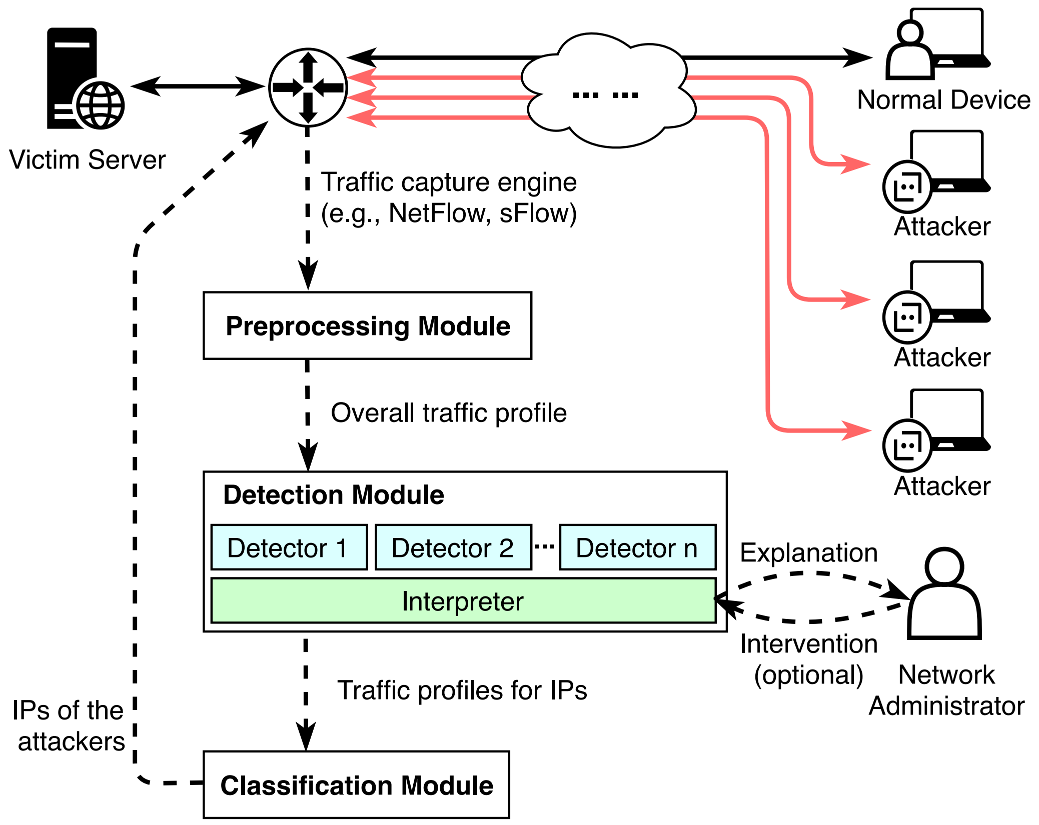


Figure 51. Operational model of the proposed approach.

assessing the network context, *traceable* for understanding the detection logic, and *heuristic* for evaluating the mitigation cost-effectiveness and formulating an appropriate plan.

5.4 Design

In our approach, DDoS detection and classification occurs at the victim end, on a vantage point that sees all the traffic to and from the victim. It can stream explanations along with detection results to the network administrator and allow interventions to the detection pipeline. Figure 51 illustrates our approach's operational model. It has three components: the preprocessing module, detection module, and classification module.

First, the preprocessing module inputs packet or flow-level traffic data from the router that runs widely used traffic capture engines, such as NetFlow [102] and sFlow [307]. It monitors the traffic in batches. Each batch is a uniform time bin, t, which is also the most basic detection unit of our approach. In our implementation, we set each batch as 5 seconds. During each batch, the preprocessing module extracts features from the input data stream to form different types of overall traffic profiles. A traffic profile can be denoted as $\boldsymbol{s}$, with $\boldsymbol{s} = \{f_1, f_2, f_3, ..., f_n\}$, where f_n denotes the value of the n-th feature during a batch t. The features in $\boldsymbol{s}$ depend on the detectors we use, as each detector may need a different traffic profile with different features.

Our approach then works in two phases: the detection phase (illustrated in Figure 52) and classification phase. In the detection phase, the detection module detects whether the network is under a DDoS attack. To provide comprehensive protection to the victim, our approach can employ multiple detectors, with each focusing on certain types of DDoS attacks. Once a DDoS attack is detected, the detection module outputs both detection results and explanations to ongoing attacks. The network administrators can review and verify the detected attack according to the explanatory information, thereby choosing to intervene in the attack defense procedure or allow our approach to automatically deal with the attack. In the end, the classification phase begins by pinpointing the IP addresses of attackers for future actions. In this phase, the classification module generates a traffic profile $\boldsymbol{p}$ for every individual IP address and classifies traffic at a fine granularity according to IP traffic profiles.

5.4.1 Detection Phase. The goal of the detection phase is to determine whether a DDoS attack is present according to the current traffic

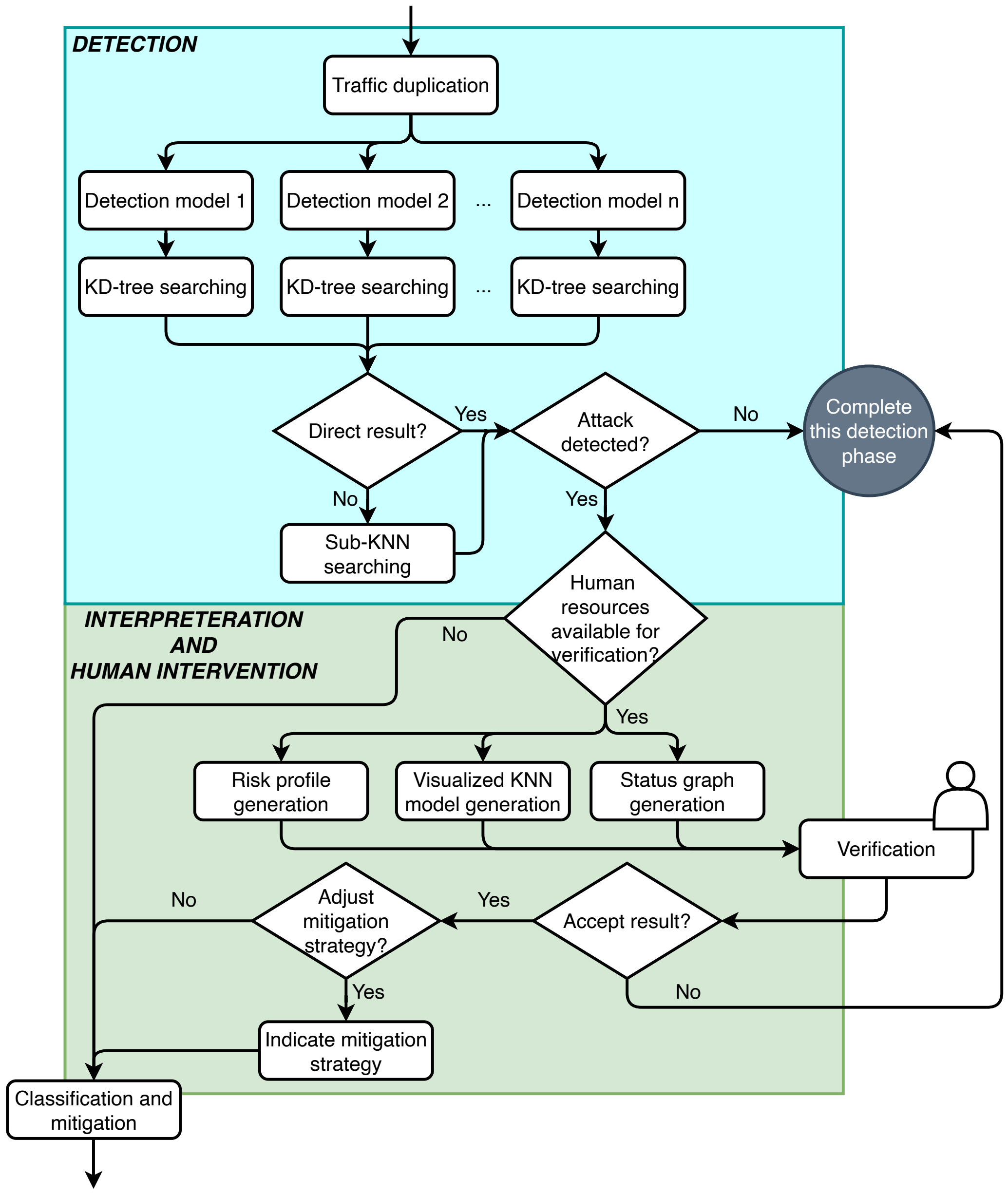


Figure 52. Workflow of the detection phase.

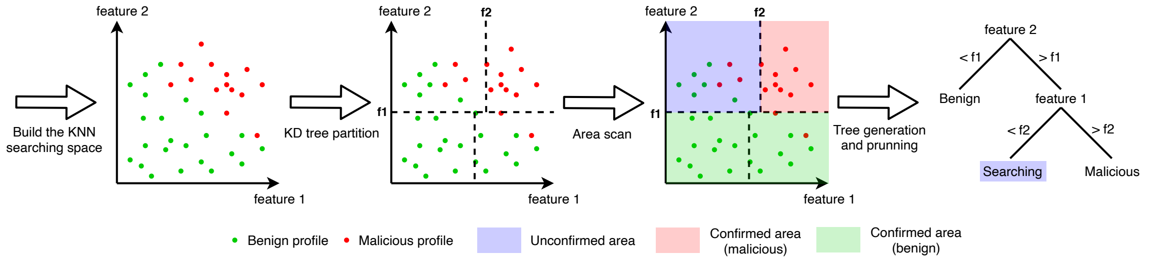


Figure 53. A DDoS detector with the modified KNN algorithm and KD tree.

profile $\boldsymbol{s}$. We use the KNN algorithm [442] to achieve the goal, as this algorithm is straightforward and reliable. The KNN algorithm is a non-parametric method used for classification, which finds the k nearest neighbors of the traffic profile $\boldsymbol{s}$ and uses their classifications to vote for the label of $\boldsymbol{s}$. Users can also choose to build multiple KNN detection models to detect a variety of DDoS attacks, as Figure 51 and 52 show.

In our implementation, we built four distinct detection models to identify TCP SYN floods, ICMP floods, UDP reflection and amplification attacks, and application-layer attacks, respectively. Each model utilizes different features and training data. The rationale behind constructing multiple KNN models to address different attacks, rather than developing a single complex KNN model, is to circumvent the curse of dimensionality [165] and overfitting. A detection model capable of handling various types of attacks generally needs to process data in high-dimensional spaces since it must encompass all the essential features of each individual attack. Nonetheless, an increase in the dataset's dimensions can render the search space sparser. Consequently, we would require significantly more training data to cover the search space; otherwise, the detection model's accuracy would be unsatisfactory. To overcome this issue, we build multiple KNN models to cover different attacks, with each model using only a few features.

Besides, users are able to adjust the voting mechanism of the KNN algorithm to get detection results with higher confidence, thereby reducing the number of false alarms in real deployments. More specifically, our approach labels the current traffic profile as malicious if more than ρ of the k nearest neighbors in the KNN searching space are malicious. We can set ρ as a number larger than 0.5 so that the detection standards will be more rigid. For example, we set ρ as 0.75 in the evaluation to eliminate the false positive rate.

However, the KNN algorithm has a notable drawback. Although the model training time is minimal, the prediction requires a time complexity of $O(nlogn)$ to complete, as it needs to enumerate the data points in the search space to find the k nearest neighbors. To address this issue, we leverage the KD tree [281] to partition the search space, thus reducing the number of data points to enumerate. With the KD tree, when an incoming traffic profile arrives, we only need to search a sub-area to predict the result. Figure 53 illustrates a simple example where only two features are included in the training and prediction process.

Furthermore, according to our experimental results, most DDoS profiles exhibit relatively large differences compared to legitimate traffic profiles. This leads to an intriguing observation that most of the search areas partitioned by the KD tree contain either benign traffic profiles or malicious traffic profiles. As shown by the red and green areas in Figure 53, we define a search area as a confirmed area if one type of traffic profile dominates the area and the number of any other type of traffic profile is smaller than ρk. If the current traffic profile $\boldsymbol{s}$ falls within a confirmed area, we can directly label the profile $\boldsymbol{s}$ with the identity of the confirmed area without conducting any KNN queries. As a result, we transform the original KNN query process into a semi-decision tree. The detection module will only trigger the search for nearest neighbors when the traffic profile $\boldsymbol{s}$ falls within an unconfirmed area. If anomalies do not occur frequently, this semi-decision tree data structure can reduce the time complexity for traffic monitoring to nearly $O(d)$, where d is the depth of the tree.

However, the use of the KD tree may lead to a slight decrease in detection accuracy. This is because the search space is partitioned into multiple sub-areas, which may result in inaccurate results when the traffic profile $\boldsymbol{s}$ falls on the boundary of two sub-areas. Nonetheless, for most DDoS attacks, legitimate traffic profiles have relatively large distances from malicious traffic profiles, leading to a significant margin between the two types of traffic profiles, thereby minimizing the impact caused by the KD tree. Moreover, by employing multiple models to detect different types of DDoS attacks, we ensure clear decision-making for each detection model, which further minimizes the impact brought by the KD tree on detection accuracy.

5.4.2 Explainability & Manual Intervention. Once our approach detects a DDoS attack, it not only outputs an alert message, but also employs an interpreter (as shown in Figure 51) to export transparent, traceable, and heuristic explanatory information to explain and quantify the attack. Such information includes a *risk profile*, a *visualized KNN model*, and a *status graph.* According to these outputs, network administrators can know the attack type, detection logic, intensity, status, confidence level of the alarm, and the cost of mitigations. Unlike some statistical approaches that provide too many metrics that can easily overwhelm network administrators, our method aims to output concise information and intuitive explanations with the help of appropriate visualizations. With a small amount of training, network administrators can understand the current situation within seconds on the basis of the interpreter's outputs, and therefore are able to quickly make manual interventions to the detection decision. Furthermore, network administrators can choose to either reject or approve the detection results. Of a particular note is that this manual intervention is optional. If the administrator does not intervene within a certain amount of time, the system will automatically execute the decisions of detectors.

5.4.2.1 Risk Profile. The risk profile Δ (where $\Delta = (m, \delta)$) is a tuple that provides the network administrators with a quantified and traceable summary about the current attack, indicating the primary cause and intensity, which meets the traceability requirement in the paradigm of explainability. Here, m is the name of the feature in the traffic profile $\boldsymbol{s}$ that primarily causes the DDoS attack. This attribute helps the network administrator determine the attack type. For example, if m is the "number of inbound ICMP packets", the victim is likely facing an ICMP attack and being overwhelmed by abundant incoming ICMP packets. δ is the smallest value by which feature f_m needs to be reduced to make the traffic profile $\boldsymbol{s}$ move to a benign position. In other words, δ is the shortest distance on f_m from the current traffic profile to a legitimate traffic profile in the KNN searching space. For example, $\Delta = (\text{"}number\ of\ inbound\ ICMP\ packets\text{"}, 8500)$ means the victim is currently under an ICMP flooding attack and we need to eliminate at least 8500 inbound ICMP packets per five seconds to mitigate the attack.

To figure out Δ for a given traffic profile $\boldsymbol{s}$ that has been labeled as DDoS attack by a detector D, we need to first find the closest benign traffic profile $\boldsymbol{l}$ in the KNN searching space of D. To achieve this, we conduct a breadth-first search. Then, we normalize $\boldsymbol{s}$ and $\boldsymbol{l}$ to ensure that features belonging to both profiles are directly comparable. In the end, we use Equation 5.1 to calculate δ and m.

$$\begin{aligned} \boldsymbol{s_\Delta} &= \boldsymbol{s}_{normalized} - \boldsymbol{l}_{normalized}, \\ m &= max(\boldsymbol{s_\Delta}).FeatureName, \\ \delta &= max(\boldsymbol{s_\Delta}). \end{aligned} \tag{5.1}$$

In a few cases, the interpreter may find multiple risk profiles from multiple detectors, which means $\Delta = \{(m_1, \delta_1), (m_2, \delta_2), ..., (m_n, \delta_n)\}$. We consider that the victim is facing either flash crowds or severe hybrid attacks under this circumstance, as the traffic volume significantly exceeds the infrastructure's capacity in multiple aspects. Here, flash crowds are large surges of legitimate traffic focusing on specific sites on the Internet over a relatively short period of time [248].

5.4.2.2 Visualized KNN Model. To fulfill the transparency requirement and provide network administrators with a clear understanding of the detection model, network context, and detection logic, the interpreter will visualize the KNN detection model in addition to the detection results. As the training and input datasets are usually of high dimensionality, the interpreter will only include three most important features of the datasets to draw a three-dimensional plot. Besides, the network administrator can choose to change the visualized features to inspect the situation from different aspects.

Such a visualized KNN model is informative. From the visualization, network administrators can observe relative distances from the current traffic profile to benign and malicious groups. According to this information, network administrators can obtain intuitive understandings regarding the detection logic, attack severity, and victim status. We further evaluate the explainability of the visualized KNN model in Section 5.5.3.

5.4.2.3 Status Graph. To facilitate rapid decision-making by network administrators based on current conditions and the cost-effectiveness of mitigation measures, the interpreter generates a status graph that provides a concise and intuitive representation of the attack stage, intensity, and confidence level of the alarm.

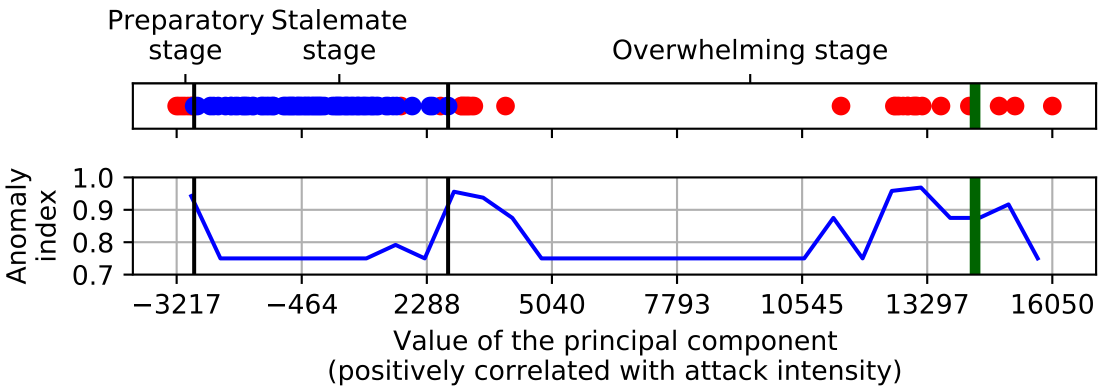


Figure 54. The status graph of a SYN flood detector. Red dots represent attack traffic profiles. Blue dots represent legitimate traffic profiles. The dark green vertical line represents the current traffic profile.

Figure 54 shows a status graph example. It consists of two subplots. The upper one uses principal component analysis (PCA) [415] to map the training and input datasets to a one-dimensional space. PCA is a technique widely used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data, while preserving as much of the data's variation as possible. More specifically, for a k-dimensional DDoS training dataset $\boldsymbol{D} \in \mathbb{R}^{N\times k}$, the interpreter uses PCA to learn a linear transformation shown in Equation 5.2.

$$\boldsymbol{T}_1 = \boldsymbol{D}\boldsymbol{W}_1,\ \boldsymbol{T}_2 \in \mathbb{R}^{N\times 1}. \tag{5.2}$$

Then, for the incoming traffic profile $\boldsymbol{s}$, we use Equation 5.3 to map it onto a two-dimensional space.

$$\boldsymbol{r} = \boldsymbol{W_1^T s},\ \boldsymbol{s} \in \mathbb{R}^{1\times k},\ \boldsymbol{r} \in \mathbb{R}^{1\times 1}. \tag{5.3}$$

In the end, our approach visualizes this one-dimensional dataset $\{\boldsymbol{r}\} \cup \boldsymbol{T}_1$, labeling DDoS traffic profiles, legitimate traffic profiles, and the input traffic profile with different colors. In other words, it is a reduced-dimensional KNN model. Network administrators can quickly learn relative spatial relationships between the current traffic profile and attack/legitimate traffic profiles from this plot.

The subplot below illustrates the anomaly index κ. This value indicates the confidence level of the detection result. The closer this value is to one, the more likely it is that the detected attack is a true positive. Since all of the alarms are detected by the KNN model, the base value of κ is equal to ρ. Then, our approach utilizes a window to move from left to right in the upper subplot, checking the number of attack and legitimate traffic profiles within the window to calculate κ.

$$\kappa = \rho + (1-\rho)\frac{n_m}{n_i + n_m}. \tag{5.4}$$

Equation 5.4 shows the calculation of the anomaly index κ, where n_m denotes the number of malicious traffic profiles within the window and n_i denotes the number of legitimate traffic profiles within the window.

In addition, by analyzing the training data, our approach divides the status graph into three stages from left to right:

– *Preparatory stage*: the attack is still in its infancy. Its intensity is low. The network administrator can choose to ignore this attack if conducting conservative defensive measures.

– *Stalemate stage*: the attack is still under the infrastructure's capacity, but it is starting to cause a noticeable impact on the network. Network administrators should mitigate the attack if conducting rigorous defensive measures. However, network administrators can still ignore this attack if they are more concerned about collateral damage caused by mitigation.

– *Overwhelming stage*: the attack is overwhelming the network, the network administrator should immediately take mitigation measures to protect the accessibility of the network.

Network administrators can know the attack status by observing which area the current traffic profile falls in.

From the example in Figure 54, we can discern from the status graph that the detected attack is in the overwhelming stage. The current traffic profile is much closer to malicious groups. Moreover, both the attack intensity and anomaly index are high. Therefore, network administrators should immediately take measures to mitigate this attack.

5.4.3 Phase Two: Classification.

The objective of the classification phase is to differentiate malicious IP addresses from benign ones, and output the malicious IPs for DDoS traffic filtering. It is important to note that the classification module will only be activated after some anomalies have been detected in the detection phase.

The design philosophy of the traffic classification is that the traffic profile $\boldsymbol{s}$ is currently in a malicious position, and we need to restrict the traffic from the most suspicious IP addresses so that the traffic profile can move to a safer position in the KNN searching space.

We begin the classification phase by building a traffic profile $\boldsymbol{p}$ for each IP address that appeared during the attack. The profile $\boldsymbol{p}$ should have the same attributes as the overall traffic profile $\boldsymbol{s}$. The only difference is that the values of features in $\boldsymbol{p}$ are calculated from the traffic of each individual IP, while the values of features in $\boldsymbol{s}$ are calculated from the overall traffic in the network. Afterwards, we sort the IP addresses in decreasing order of the risk degree, where the risk degree is a number that indicates how suspicious an IP is. According to the risk profile Δ ($\Delta = (m, \delta)$) we obtained from the DDoS detection phase, we define the risk degree of an IP address as $f_m^{(\boldsymbol{p})}$. Finally, we conduct traffic filtering on IP addresses such that the overall traffic profile can move to a benign area.

However, legitimate clients may sometimes have significant risk degrees as well. Classifying the IP addresses only according to the risk degree may cause significant collateral damage. To address this issue, we also need to minimize the impact on other features of the overall traffic profile $\boldsymbol{s}$ when determining the malicious traffic sources. We consider this as an optimization problem with two constraints, which can be demonstrated by Equation 5.5. Here, G denotes the complete set of IP addresses we have seen in the network during the DDoS attack, G_m denotes the set of malicious IP addresses that the classification program will output for future actions, and $p^{(i)}$ denotes the traffic profile of the ith-IP.

$$\begin{aligned}\underset{G_m}{\operatorname{argmax}} f(G, G_m) &= \sum_{g \in G, g \notin G_m} \sum_{i \in g} \left\| \boldsymbol{p}^{(i)} \right\|_2 \\ &= \sum_{g \in G, g \notin G_m} \sum_{i \in g} \sqrt{\sum_{k=1}^{n} \left| f_k^{\boldsymbol{p}^{(i)}} \right|^2},\end{aligned} \tag{5.5}$$

$$\begin{aligned}\text{subject to: } & \sum_{g \in G_m} \sum_{i \in g} p_m^i \geq \delta, \\ & G_m \subseteq G.\end{aligned} \tag{5.6}$$

Equation 5.6 shows two constraints: (1) after eliminating all the traffic from malicious IP addresses (set G_m), the overall traffic profile should be reduced by at least δ on f_m in the KNN searching space; (2) the malicious IP set G_m should be a subset of the complete IP set G.

Algorithm 3 Recognition of malicious IPs with grid sorting

```
1: input: risk profile $\Delta = (m, \delta)$
2: input: complete IP set $G$
3: initialize set $G_m$ to store the malicious IP addresses
4: grid partitioning: $G = \{g_1, g_2, g_3, ..., g_n\}$
5: $G.sort()$    ▷ in decreasing order of feature $m$ and increasing order of other features
6: for $g$ in $G$ do
7:     $G_m.add(g.items())$
8:     $val \longleftarrow \sum_{i \in g} f_m^{p^{(i)}}$
9:     $total\_eliminated \longleftarrow val + total\_eliminated$
10:     if $total\_eliminated >= \delta$ then
11:         return $G_m$
12:     end if
13: end for
```

Deriving the optimal solution of this optimization problem is expensive, especially when the network we are monitoring is at the ISP-level. Hence, we designed Algorithm 3 to obtain a near-optimal solution G_m efficiently. Since the time complexity of sorting the IPs according to the risk degree is $O(nlogn)$, the algorithm conducts the grid partitioning on the searching space to accelerate the IP classification. Then, we need to eliminate IP addresses along the m axis and minimize impacts on other features at the same time. With this grid configuration, we can always find a corner grid g_m that has the largest value on feature m but also has the smallest values on irrelevant features. The classifier considers the grid g_m as the most suspicious grid and gives it the highest priority in classification. Afterwards, the algorithm sorts the remaining grids in decreasing order of feature

m and increasing order of other features. Finally, the algorithm eliminates IPs grid-by-grid in such order until the overall traffic profile returns to the benign area. Figure 55 illustrates an example of such procedure.

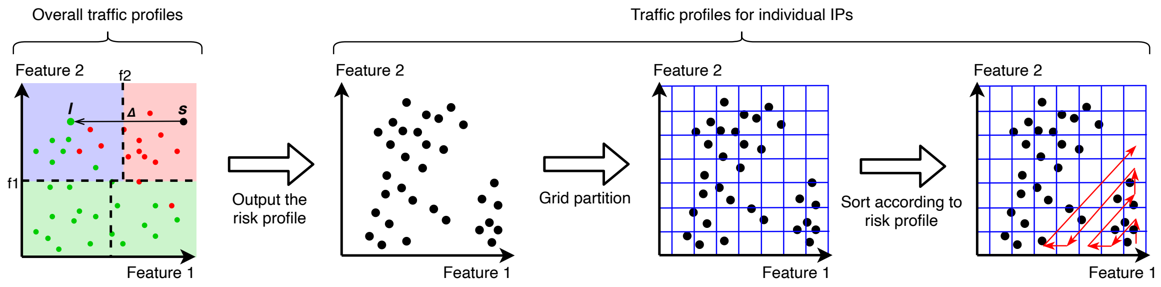


Figure 55. An example of the classification process, where the classification module reads the risk profile, partitions the searching space, and find the malicious IP set G_m grid-by-grid.

5.4.4 Adaptability.

The proposed approach offers superior adaptability compared to other learning-based methods. When deploying a pre-trained detection model in a new network environment, users are not required to retrain the model for a suitable fit. Instead, they can leverage a variety of prior knowledge to evolve the model, thereby enhancing its robustness across different environments.

Here, we assume the user will have some type of limited information about the new network environment as prior knowledge. Such information includes the network traffic measurement results or link bandwidth information about the network environment, some training samples for online learning, and incomplete threshold values for DDoS detection. Any type of the above information can evolve the detection model and help the model adapt to the new environment.

5.4.4.1 Mapping via Traffic Measurement.

Assuming that we have the network traffic measurement results about the new network environment, we can normalize the KNN searching space from the trained environment to the new environment according to the traffic distributions of the two networks. The easiest way to do this is by using min-max normalization for the conversion process.

$$\begin{aligned} l &= max(\boldsymbol{D}_{new}[:, i]) - min(\boldsymbol{D}_{new}[:, i]) \\ \widehat{\boldsymbol{D}}[\boldsymbol{:}, \boldsymbol{i}] &= l \cdot \frac{\boldsymbol{D}[:, i] - min(\boldsymbol{D}[:, i])}{max(\boldsymbol{D}[:, i]) - min(\boldsymbol{D}[:, i])} \end{aligned} \tag{5.7}$$

Equation 5.7 shows the conversion process, where $\boldsymbol{D}$ denotes the original training dataset and $\boldsymbol{D}_{new}$ denotes the sampled traffic from the new network environment. By mapping the original training data to the new network environment, our approach is able to conduct DDoS detection without retraining or re-collecting any new training data.

5.4.4.2 Online Updating for KD-tree. If the traffic monitoring system can obtain labeled traffic with the system running, we can conduct online learning on the proposed detection model, thus making it gradually fit a new environment. The KNN algorithm does not require training, making it very suitable and efficient to conduct online learning. However, the KD-tree, along with the confirmed areas, needs to refresh to reflect new knowledge. We can control the program to update the classifier only during the idle times to reduce the performance impact on the detection system. Nevertheless, the time complexity of refreshing the model is only O(n).

5.4.4.3 Integration with Existing Thresholds/Rules. In certain situations, network administrators may already have imperfect detection rules (e.g., threshold-based rules) tailored to their network environment. Below are a few examples of such rules:

```
if (traffic.packets_per_second > 2_000_000
    or traffic.kbs_per_second > 1_800_000
    or traffic.in_out_ratio > 80
```

Algorithm 4 Integration with existing rules

```
1: input: existing rule table $T$ as a stack
2: input: detection model $D$                                   ▷ $D$ is a semi-decision tree
3: while $T$ is not empty do
4:     $r \longleftarrow T.pop()$
5:     if $D(r.condition)$ exists and overlaps with searching area set $S$ then
6:         remove overlapped areas from $S$
7:         $T.push(r)$
8:     else if $D(r.condition)$ exists then
9:         $D.update(r)$
10:    else                                                     ▷ $D(r.condition)$ not exists
11:        $D.root.rightChild \longleftarrow D$          ▷ right child will be called when not satisfying the condition
12:        $D.root \longleftarrow r.condition$
13:        $D.root.leftChild \longleftarrow FILTER$ action
14:    end if
15: end while
16: return $D$
```

```
    or traffic.external_ips > 15_000):
    alert()
else:
    pass
```

Network administrators can integrate our approach with existing rules to enhance DDoS protection efficacy without disrupting the current detection logic or significantly increasing the rule budgets. Since the pre-trained DDoS detection model is a semi-decision tree, users can incorporate existing detection rules into the pre-trained model by modifying the tree structure. This design allows our detection approach to adapt to existing knowledge without substantially increasing rule budgets and detection overhead. Algorithm 4 illustrates an example procedure for integration, where the existing detection rules have higher priority. Users can also specify different decision priorities based on the current situation.

Table 20. Datasets for training and testing.

Dataset Name	Format	Size	Attack Type	Background Traffic	Used For
DARPA 2009 DDoS [13]	pcap	1.09 GB	TCP SYN flood attack	✓	Training & Testing
CAIDA 2007 DDoS [12]	pcap	12.08 GB	ICMP flood attack		Training & Testing
FRGP NTP Flow Data [15]	Argus flows	1.60 TB	NTP reflection attack	✓	Training & Testing
DDoS Chargen 2016 [18]	flow-tools	74.05 GB	UDP reflection and amplification attacks	✓	Training & Testing
FRGP Colorado Traffic [5]	FlowRide & NetFlow	$>$ 5.00 TB	Various	✓	Testing

5.5 Evaluation

In this section, we assess our approach from various perspectives. We tested our approach not only in simulated environments using multiple publicly-available DDoS datasets (Subsection 5.5.2), but also deployed it at FRGP [5], a regional IXP in Colorado State, to examine its adaptability and usability in real-world scenarios (Subsection 5.5.4). Additionally, we conducted a questionnaire survey to quantitatively evaluate the explainability of our approach (Subsection 5.5.3).

5.5.1 Features & Training Data. Our learning-based approach requires labeled training data as input in order to build the detectors for each attack type. Therefore, we picked several representative DDoS datasets from public repositories and captured traffic in real-world environments to train and test our approach. Table 20 shows the public datasets we used and the types of attacks they contain. These datasets and our captured traffic cover volumetric attacks (e.g., ICMP flood, UDP reflection and amplification attacks), protocol attacks (e.g., TCP SYN flood attacks), and application-layer attacks (e.g., HTTP flood, Slowloris, etc.). We separately trained four DDoS detection models using the datasets, with one dedicated to TCP SYN flood, another to ICMP flood, a third one for UDP reflection and amplification attacks, and a final one for application-layer attacks. Together, these models can provide comprehensive protection to the victim server.

Table 21. Features we utilize for detecting and classifying different categories of DDoS attacks.

Attack Type	**Features We Use**
TCP SYN flood — *protocol attack*	# of inbound TCP packets / # of outbound TCP packets, # of TCP packets, # of inbound SYN packets, # of outbound ACK packets, # of inbound ACK packets
ICMP flood — *volumetric attack*	# of inbound ICMP packets / # of outbound ICMP packets, # of ICMP packets, # of inbound echo requests, # of outbound replies (destination unreachable)
UDP reflection & amplification attack — *volumetric attack*	# of inbound UDP bytes / # of outbound UDP bytes, # of UDP bytes, # of inbound UDP packets / # of outbound UDP packets, # of UDP packets
HTTP GET flood, Slowloris, DNS query attack, etc. — *application-layer attack*	# of inbound bytes / # of outbound bytes, # of bytes, # of sessions, # of inbound packets / # of outbound packets, # of packets, avg packet interval

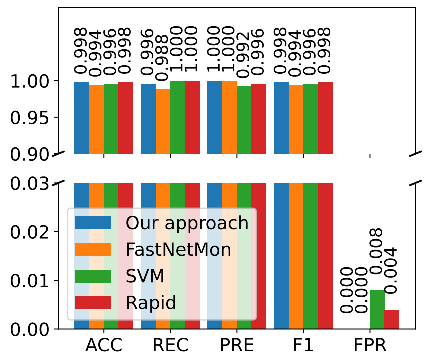
The training datasets come in various formats, ranging from packet-level pcap data to flow-level connection data. Since our approach operates at the flow level, we preprocess the data by converting the original datasets into traffic profiles tailored to different detection models with a granularity of five seconds. We also sampled a small portion (approximately 10%) of the data from the DDoS datasets for our testing datasets. These testing datasets were not used during model training but were instead utilized in the testing phase. Moreover, we sampled network traffic from a router at FRGP to simulate legitimate background traffic, thereby complementing the dataset. The overall ratio of DDoS training data to legitimate background training data is 1:2.

As our approach works best with low-dimensional datasets, we selected the best feature sets based on univariate statistical tests. More specifically, we performed χ^2 tests to the data samples to retrieve only 4-6 best features. Table 21 enumerates the four sets of features we selected to train the four different DDoS detectors. The most frequently used feature was the ratio of the inbound traffic volume to the outbound traffic volume. We found that the features listed in Table 21 are useful in identifying the majority of DDoS attacks.

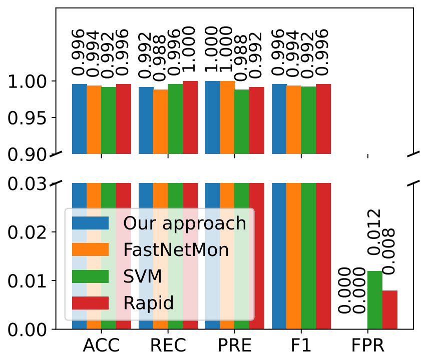
5.5.2 Detection & Classification Efficacy. To evaluate the detection and classification efficacy of our approach, we first built a simulation environment where a virtual switch continuously streams collected traffic to the proposed system. Such a simulation environment enables us to conduct convenient and efficient tests. During the evaluation, we simultaneously replayed legitimate traffic and a portion of the DDoS test traffic. We also dynamically adjusted the traffic volume during the test to effectively mimic real-world DDoS scenarios.

5.5.2.1 Detection Efficacy. For comparison tests, we utilized three additional DDoS detection approaches. One is a DDoS detection model based on a support vector machine (SVM)[288]. We trained this model using the same training data and features as shown in Table21. Another is FastNetMon [289], an open-source commercial DDoS detection program. This threshold-based DDoS detection approach is widely employed in small to mid-sized enterprises due to its high efficiency and accuracy. Lastly, we included Rapid [270], a hybrid DDoS detection method that combines LSTM and multi-layer perceptron. The test dataset consists of at least 250 episodes of legitimate traffic traces and at least 250 episodes of traffic traces with attacks. An episode is the most basic detection unit, containing more than five seconds of replayed network traffic.

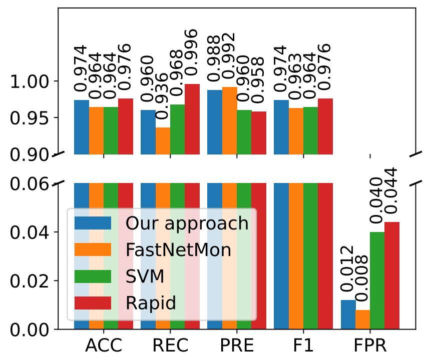
Figure 56 illustrates the comparison results for DDoS detection under the simulated environments. For both SYN flood and ICMP flood attacks, all the three approaches can achieve very decent detection efficacy. As for UDP and application-layer attacks, although Rapid and the SVM-based approach are slightly superior to our approach in terms of recall scores, they perform worse in terms of the false positive rates. A low false positive rate is essential for the detection system's usability in real-world deployments, as a high number of false alarms will either cause too much collateral damage or force network administrators to ignore the detection results. Thus, when accuracies are similar, users tend to choose the



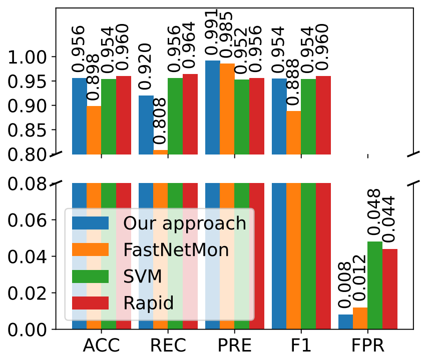
(a) SYN flood.



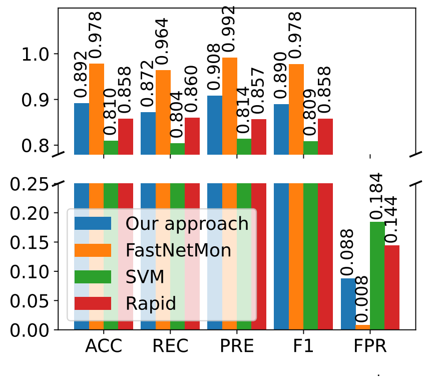
(b) ICMP flood.



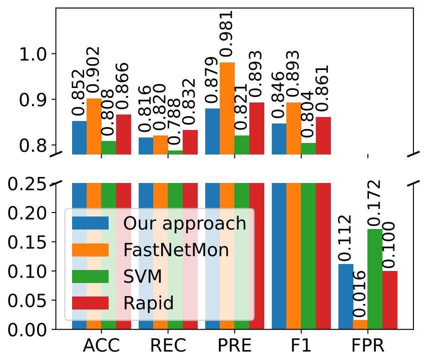
(c) UDP flood.



(d) Application-layer DDoS.



(e) Volumetric and protocol DDoS (unbalanced training data).



(f) Application-layer DDoS (unbalanced training data).

Figure 56. Comparison results of DDoS detection efficacy (ACC: Accuracy, REC: Recall, PRE: Precision, F1: F1 score, and FPR: False positive rate).

approach with a significantly lower false positive rate. Compared with FastNetMon, our approach has a similar false positive rate. However, our approach is superior to FastNetMon in terms of accuracy and recall score.

We also halved the training data, resulting in a 1:4 ratio between the DDoS training data and legitimate background training data, to assess the detection efficacy in the presence of unbalanced and insufficient training data. Figure 56e and 56f illustrate the results. We can see that when the training data is unbalanced, FastNetMon works significantly better than the other approaches, as it is a threshold-based approach. Among the other three approaches, our method demonstrates superior detection efficacy compared to the SVM-based approach and exhibits comparable efficacy to Rapid.

5.5.2.2 Classification Efficacy. As for the traffic classification, we first replayed a collected network traffic dataset in a Mininet-based [7] network environment. This dataset consists of 25 minutes of network traffic with both DDoS attack and legitimate packets. Then, we ran FastNetMon and the proposed approach respectively, conducting mitigation on malicious IP addresses reported by them throughout each process. Simultaneously, we observed the network situation to evaluate the classification efficacy. To ensure a fair procedure, we did not intervene in the detection process during evaluation.

Figure 57 shows the classification efficacy results, where the y-axis indicates the number of packets. By mitigating all the traffic from the attackers classified by the two approaches, we can see our approach can eliminate more malicious traffic than FastNetMon. The only drawback of our approach is that the classification will only be triggered when an attack is detected. After proceeding with mitigation, once the traffic profile is no longer labeled as malicious, our approach will stop

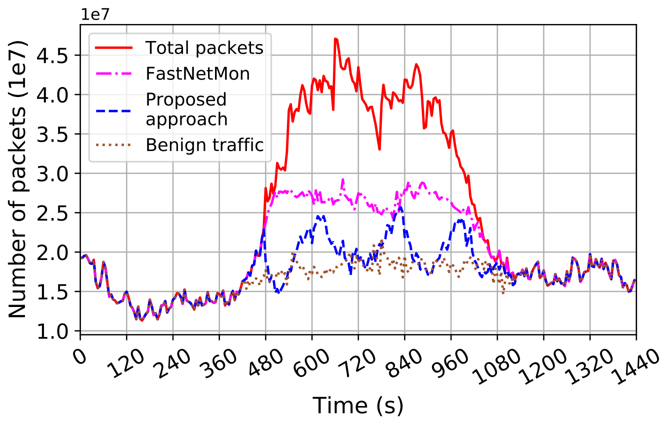


Figure 57. Efficacy of DDoS traffic classification.

classifying IP addresses as malicious, and only begin classification again as soon as the traffic profile is labeled malicious again. This explains the periodic fluctuations on the number of packets for our approach as seen in the figure.

5.5.2.3 Timeliness. We measured the runtime of our approach on a 100 Gbps link (please refer to Section 5.5.4 for details of the link) and presented the results in Figure 58. This figure shows three cumulative step histograms, illustrating the runtime for pre-processing a batch of traffic (five seconds), monitoring a batch of legitimate traffic, and monitoring a batch of traffic with attacks, respectively. Here, the runtime for monitoring legitimate traffic consists of the time consumption of pre-processing and detection; the runtime for monitoring traffic with attacks consists of the time consumption of pre-processing, detection, and classification.

From the figure, we can see that the runtime is short when there are no attacks present, considering that the program has a five-second time window to operate. Moreover, as the detection model is a semi-decision tree, it will directly output the results without conducting any KNN queries if the traffic profile is situated in a confirmed benign area. Thus, the majority of time is spent on

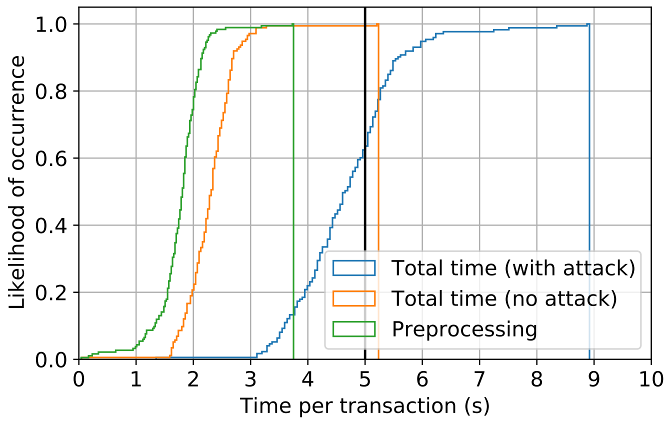


Figure 58. Cumulative step histograms of processing time (tested with 100 Gbps flow-level traffic).

traffic pre-processing when monitoring only legitimate traffic. When there is a considerable amount of incoming DDoS traffic, the runtime almost doubles since fine-grained IP classification is time-consuming. Fortunately, when an attack is detected, the system does not need to complete the calculation within five seconds to catch the next batch. The top priority at the time an attack is detected is to mitigate the attack, and therefore, an increased delay in classification is still acceptable.

In conclusion, our approach is efficient when detecting and classifying DDoS traffic. With delays of around two seconds during idle time and five seconds during the DDoS peak, our approach is able to produce timely defense for the victim.

5.5.3 Explainability. To evaluate the explainability of our approach, we conducted a questionnaire survey, which is a formal and effective method in Human-Computer Interaction (HCI) research [407], to collect feedback from participants and assess their understanding of the system's functionality and decision-making process.

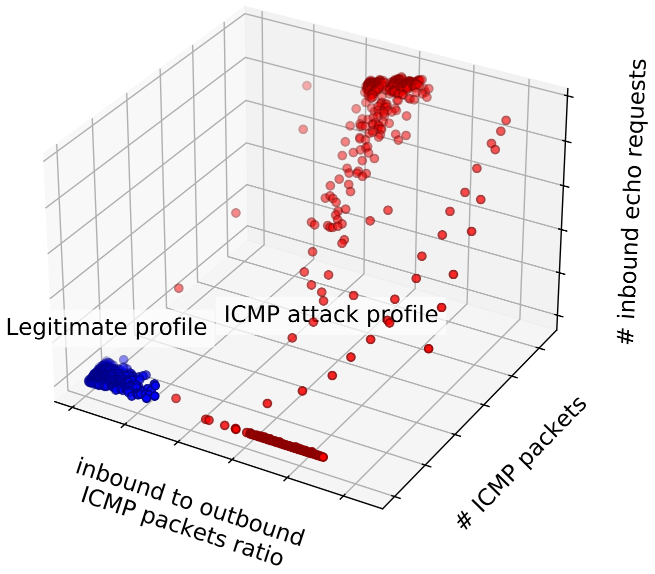
Composition of the participants	
Participants with DDoS-related expertise	15
Participants with security backgrounds but not DDoS-related expertise	4
Participants without security backgrounds	4
The total number of participants	23
Basic information of the questionnaire survey	
Number of questions	25
Approximate time to explain the usage of our approach (min)	15
Approximate time to complete the survey (min)	30

Table 22. Basic information of the questionnaire survey.

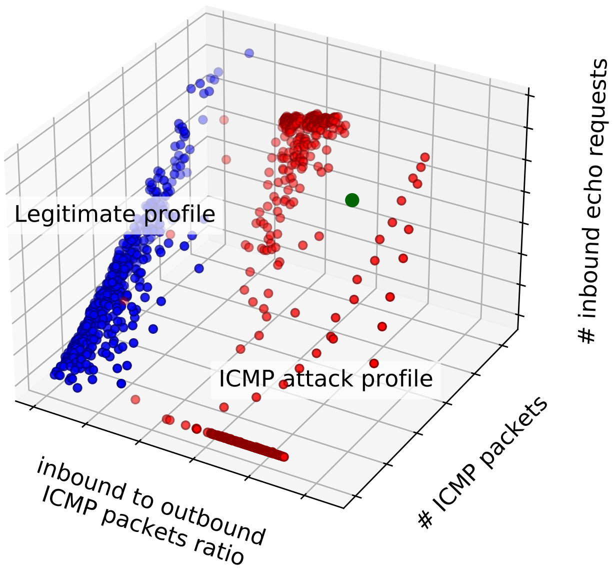
We disseminated survey questionnaires to a range of security labs and individuals without a security background in the USA and China. In total, 23 people participated in the survey. Table 22 provides an overview of the participants' basic information as well as essential details about the questionnaire survey.

Before the questionnaire, we provided a brief introduction to the background knowledge, our design, and the output explanatory information. We then presented several examples of the outputs to demonstrate how they explain detected attacks and how to interpret them. Figure 59 and 60 display a few examples from the questionnaire.

We proceeded to ask participants questions about the explainability of our approach, such as the ease of understanding the outputs, the intuitiveness of the visualizations, and whether the explanatory information met the design objective. Finally, we administered tests to assess participants' comprehension of the explanatory information for various attack types and their ability to make correct interventions. Specifically, we presented scenarios of different attack types detected by our approach and asked participants to interpret the explanatory

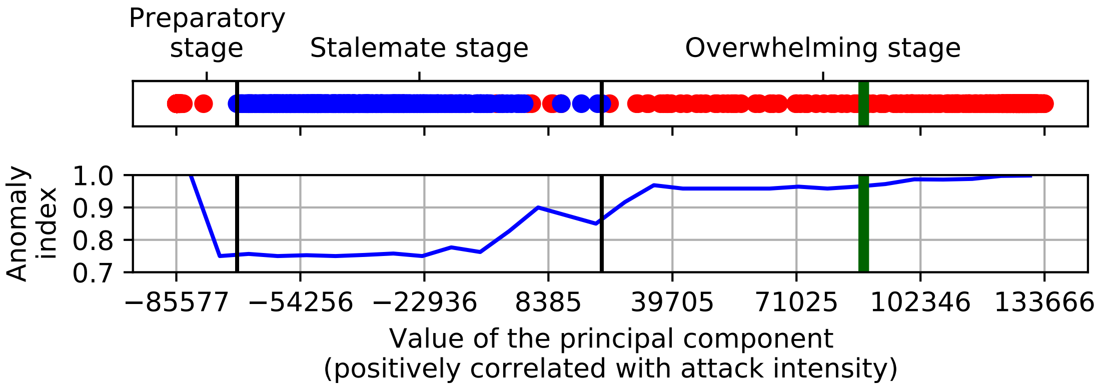


(a) Before normalization. We can clearly see that the training data does not fit the network environment.

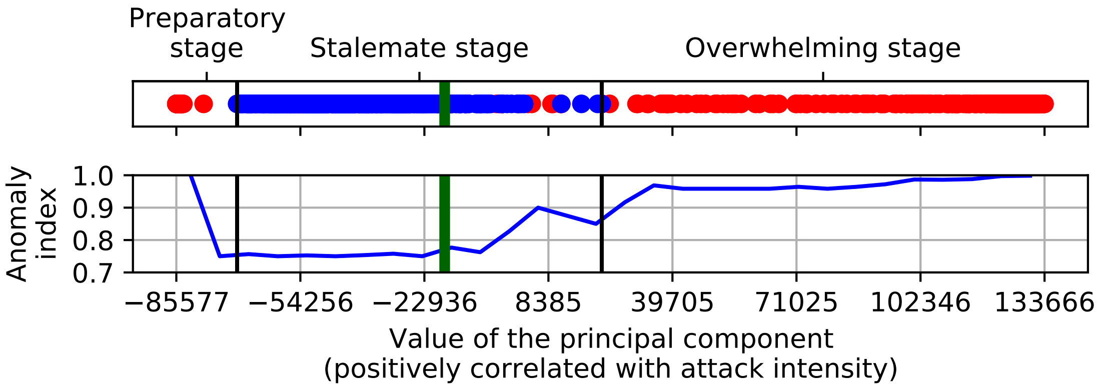


(b) Normalized according to the traffic distribution. The dark green dot is the current traffic profile for inference, whose risk profile $\Delta = ("number\ of\ inbound\ ICMP\ packets", 52041)$.

Figure 59. Examples of visualized KNN detection models for identifying ICMP attacks.



(a) Status graph of an attack shown in Figure 59b. Network administrators should proceed with mitigation as this attack has a high intensity and is already in the overwhelming stage.



(b) A detected ICMP flooding attack. This attack is in the stalemate stage. Network administrators can either ignore this attack if following a conservative protection policy or proceed with mitigation immediately if following a more aggressive protection policy.

Figure 60. Examples of status graphs for explaining ICMP attacks.

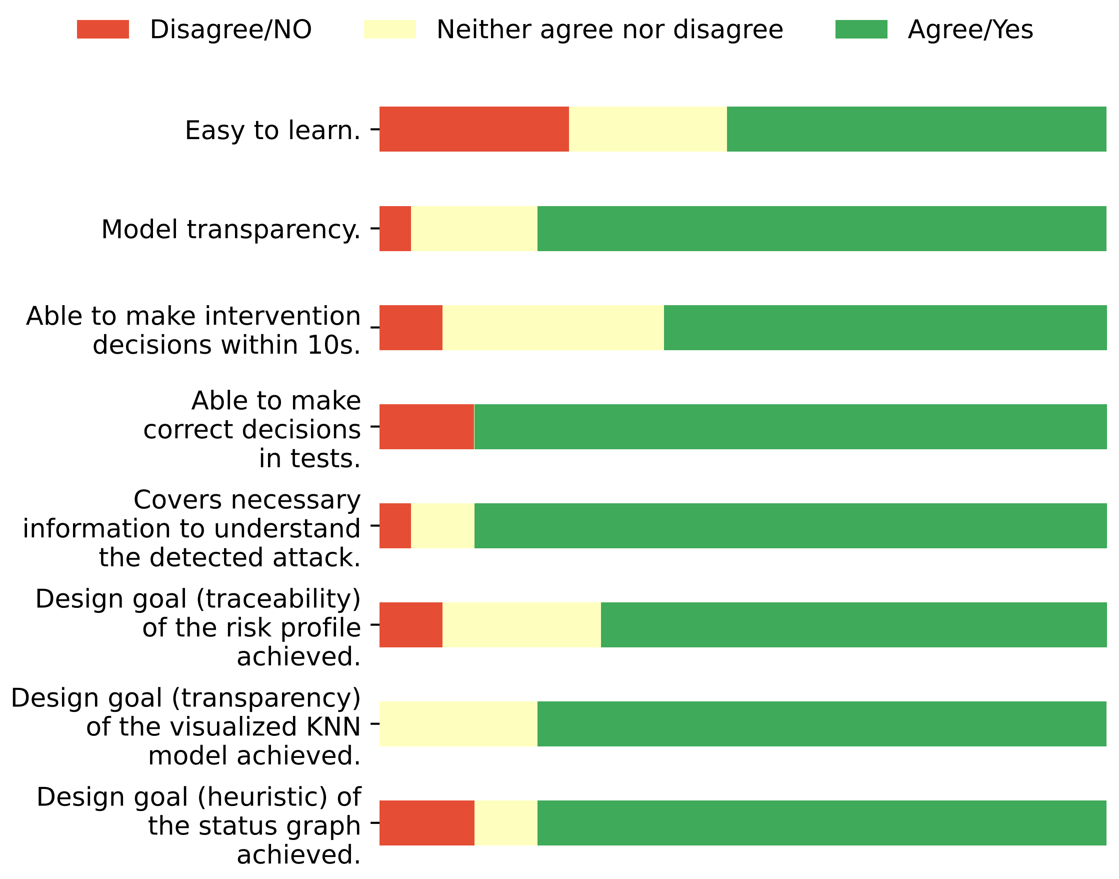


Figure 61. Selected questionnaire evaluation results on the explainability of our approach.

information and recommend intervention measures for the next step under varying circumstances. Responses were collected anonymously to protect privacy and minimize bias.

Figure 61 presents some key findings from our questionnaire evaluation. Although a few individuals questioned the explainability aspects of our approach, the majority of participants agreed that the risk profile helps users understand the root cause and quantify the attack. Additionally, the visualized KNN model provides an intuitive explanation of the network context, detection models, and detection logic for network administrators. The status graph illustrates the current attack stage, intensity, confidence level of the alarm, and mitigation cost-effectiveness, ultimately guiding network administrators in making appropriate interventions. Therefore, in terms of transparency, traceability, heuristic, and ease of learning, our proposed approach successfully achieves its design goals.

5.5.4 Case Study: A Real-world Deployment.

In addition to the evaluation under emulation environments, we deployed our approach at several links in FRGP to further test its deployability and adaptability. This real-world deployment also provides a good opportunity to demonstrate how explanatory information can help network administrators adopt conservative tactics for eliminating false alarms.

5.5.4.1 Measures for Ethical Considerations.

As the network traffic from FRGP contains private information of users and trade secrets of operators, we take effective measures to address possible ethical considerations. Data is collected by FRGP operators and their collaborators from a local educational institution on an ongoing basis. We formulate a Memorandum of Agreement (MoA) with FRGP operators and their collaborators to stipulate the correct usage and accessibility of the data. To protect the privacy of users and prevent data leakage, we set rigorous regulations for data analysis and storage. We list the regulations below:

1. The IP addresses in the network traffic data are anonymized in a prefix-preserving manner with CryptoPAN [425], before collection and storage. This ensures we cannot trace back to individual users during our deployment.

2. All the data is stored on a restricted server. Besides the SSH port, all the ports on the server are closed, and no connections can be initiated from the server. This minimizes the risk of accidental leakage of network data.

3. Our approach can only be deployed on the restricted server.

4. We are only allowed to receive the detection results from the restricted server. Other information related to the IXP operations have to remain on the restricted server, such as prefix-level measurements, the trained model, and pre-processed data.

5.5.4.2 Deployment Setup. The restricted server where our approach is deployed has an Intel Xeon Silver 4116 processor with 64 GB of RAM. The flow-level data is collected from multiple routers at FRGP during a 3-month period between 10:20 MST on February 24 to 21:40 MST on May 21, 2020. At its peak, the traffic volume usually reaches 100 Gbps during the day. Our approach can simultaneously obtain access to network traffic flows in three formats, which are NetFlow, Argus Flow [3], and FlowRide, a newly developed flow-capture tool that summarizes traffic every five seconds. The pre-processing module converts the traffic flows into the overall traffic profiles and IP-level traffic profiles for each detector.

As was true in the evaluation of the simulation environment, the deployed detection model was pre-trained with datasets shown in Table 20. To adapt the pre-trained detection model to the FRGP environment, we conducted several measurements on the network to obtain the data distribution for each traffic feature used by the detectors. Then, we mapped the pre-trained model to the FRGP environment according to these distributions. While the program was running, we were able to receive the detection results for different types of attacks (i.e.NTP, TCP SYN, ICMP, and UDP attacks). During the evaluation, the FRGP operators also gave us information about DDoS attacks they discovered using Arbor Network's PeakFlow and Threat Mitigation System (TMS) [1]. Of a particular note

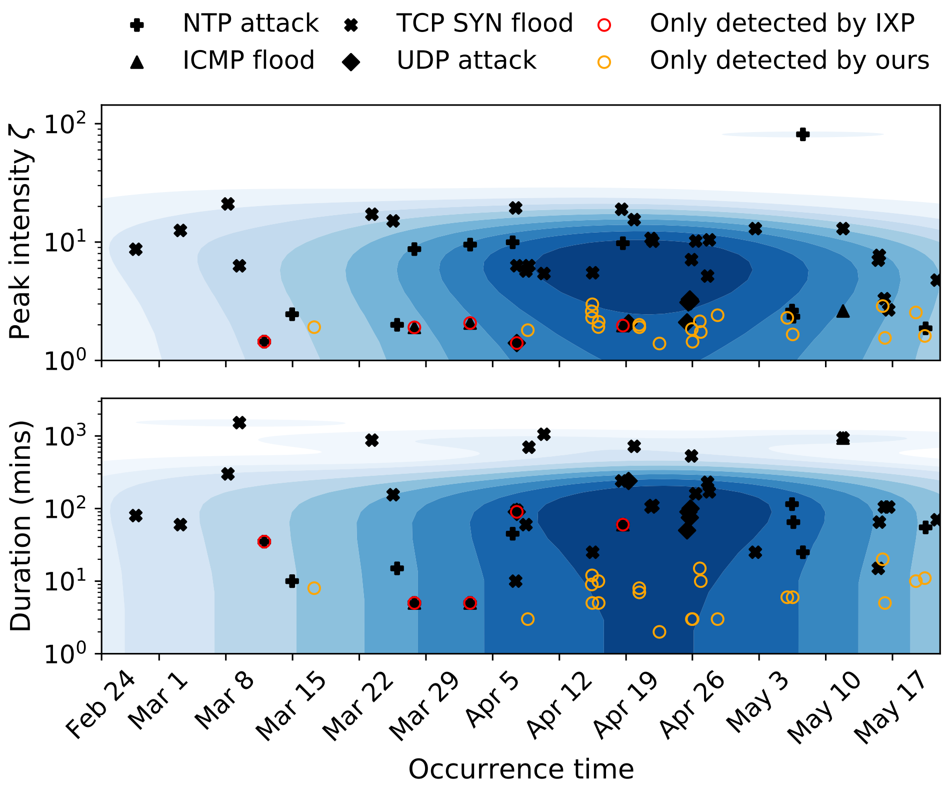


Figure 62. Detection results from 10:20 MST on February 24 to 21:40 MST on May 21, 2020. Red circles indicate the attacks only reported by FRGP operators but not detected by our approach. Yellow circles indicate the attacks only detected by our approach but not reported by FRGP operators. Other dots indicate the attacks detected by both parties. The depth of the background color represents the density of attacks.

is that the attacks reported by FRGP cannot represent ground truth as the IXP also suffers from false positives and false negatives, but they have good reference values for evaluating our approach. In addition, our contract with FRGP operators does not allow us to alter any traffic flows in their network, so we did not evaluate the classification efficacy in this deployment.

5.5.4.3 Findings. To better quantify the traffic change during an anomaly, we define **peak intensity index** ζ, calculated as $\zeta = V_{peak}/V_{exp}$, where V_{peak} denotes the peak volume of the anomaly and V_{exp} denotes the expected traffic volume. For an anomaly with a short duration (less than 30 minutes), we treat the traffic volume right before the anomaly as V_{exp}. For an anomaly with a longer duration, we calculate V_{exp} by statistically averaging legitimate traffic volumes at the same time in the surrounding seven days. Figure 62 shows the anomaly detection results. The top subplot illustrates the peak intensity indexes ζ of the anomalies occurring at different times. The bottom subplot illustrates the duration of the detected anomalies at different times.

Our approach successfully detected over 90% of DDoS attacks reported by FRGP operators, including all severe attacks with a ζ greater than 2. The five missed alarms (highlighted with red circles in Figure 62) were all low-intensity attacks that did not significantly damage the systems.

Furthermore, our approach proved more sensitive in detecting DDoS attacks, generating 21 alarms that FRGP operators missed (highlighted with yellow circles in Figure 62). These 21 alarms involved low-intensity, short-duration attacks, which could represent small-scale floods undetected by FRGP's system or false positives. The effective explainability of our approach enabled network administrators to determine that most of these attacks were in preparatory or stalemate stages based on their status graphs. Consequently, if network administrators opt for a conservative mitigation policy, they can quickly review the explanatory information and choose to disregard these alarms.

In conclusion, the real-world deployment demonstrates the adaptability and usability of our approach. Besides, the explanatory information can quickly help the network administrators identify possible false positives or less threatening attacks, thereby making necessary interventions.

5.6 Discussions

In this section, we explore the limitations and open Issues tied to our proposed methodology. Additionally, we deliberate on the potential for broadening the application of the proposed methodology.

5.6.1 Limitations and Open Issues. A primary contribution of this approach is to offer a learning-based DDoS detection solution that features a high level of explainability and adaptability. However, this approach has the following limitations:

– As a learning-based approach, our method may not be able to tackle zero-day DDoS attacks. For example, during the FRGP deployment, we missed a SYN-ACK distributed reflection denial of service (DRDoS) attack that targeted a stateful firewall at the end of April. The characteristics of this attack were not contained in the training data, making our approach difficult to capture it.

– Due to the characteristic of KNN algorithm, our approach may be vulnerable to certain adversarial attacks. We can utilize some adversarial machine learning techniques such as data smoothing to fix this problem, but it will inevitably decrease detection accuracy.

In addition, our approach faces several open issues as possible future working items:

– We can leverage other explainable machine learning algorithms such as random forests to enhance our approach. Furthermore, it is meaningful to compare the current algorithm with random forests.

- More features may be explored to improve the accuracy for detecting certain complicated DDoS attacks.

- We can further enhance the evaluation of this work by enhancing the efficacy evaluation of our traffic classification algorithm.

5.6.2 Methodology Generalization. Some concepts, techniques, and methodologies introduced in this chapter hold applicability beyond DDoS detection, FGTA tasks or even disparate domains.

5.6.2.1 Requirements for explainability. This chapter presents a collection of requirements——transparency, traceability, and heuristic——that are tailor-made for DDoS detection explainability. While these specifications may not be universally applicable across all machine learning tasks, they could serve as a benchmark in crafting explainable detection systems for various other forms of attacks, anomalies, or intrusions.

5.6.2.2 Explainability designs. We have proposed a design for explainability that adds an additional layer atop an existing machine learning model to yield interpretable outcomes. While this specific design is currently tailored for the modified KNN algorithm we proposed, the underlying concept could potentially be adapted to other scenarios or machine learning algorithms.

5.6.2.3 Adaptability designs. In our methodology, we employ an array of techniques to customize the detection system for different network environments. These include model mapping, online updating for KD-tree, and integration of existing thresholds/rules. These strategies could potentially be utilized in other FGTA tasks that implement KNN models or necessitate integration with pre-existing knowledge.

5.7 Conclusions

In this chapter, we exemplify the enhancement of FGTA's explainability and adaptability, making it more practical and applicable in real-world situations. Specifically, we introduce a learning-based methodology for the detection and classification of DDoS traffic, serving as a tangible example of these improved FGTA features.

Compared with the existing approaches, the proposed method offers (1) explainability and (2) adaptability. With the KD tree and the modified KNN algorithm, this approach generates a tree-like classifier, which not only makes predictions fast but also provides network administrators with a clear perspective of network conditions, detection logic, attack stages, and mitigation costs. Furthermore, people can easily adapt the detection model to a different environment by utilizing prior knowledge *without* retraining the model from scratch. Benefiting from grid sorting, the classification module can reduce collateral damage to a large extent and generate the results promptly.

We trained the detection model with representative DDoS datasets from public repositories in our simulation environment. We then evaluated this approach in both simulated environments and a real-world setting. The evaluation results show the efficacy and efficiency of this approach in both settings, as well as its adaptability from the small simulated environments to a real IXP setting. As for explainability, the questionnaire evaluation indicates that in terms of transparency, traceability, heuristic, and ease of learning, our method achieves the expected design goals.

CHAPTER VI

APPLICATION-LAYER DDOS DEFENSE WITH REINFORCEMENT LEARNING

In earlier chapters, we examined FGTA's capabilities in extracting fine-grained knowledge from network traffic and identifying complex malicious activities. Despite this, FGTA is not optimal for detecting specific application-layer malicious activities (e.g., L7 DDoS) due to its restricted application-layer situational awareness. Undoubtedly, endpoints possess the most comprehensive knowledge about application-layer activities. Consequently, in this chapter, we investigate the joint use of endpoint-based defense and FGTA to effectively combat L7 DDoS attacks.

In this chapter, we propose a novel reinforcement-learning-based approach for defending against L7 DDoS attacks. We introduce a multi-objective reward function that guides a reinforcement learning agent in determining the most suitable action for mitigating L7 DDoS attacks by incorporating both network traffic information and endpoint-side operational data. Consequently, the agent can apply various strategies under different conditions while actively monitoring and analyzing the victim server to provide protection. During an overwhelming L7 DDoS attack, the agent will aggressively mitigate as many malicious requests as possible, ensuring the victim server remains operational (even if it means sacrificing a small number of legitimate requests). In less critical situations, the agent will adopt a more conservative approach to mitigating malicious requests, prioritizing the minimization of collateral damage to legitimate requests.

This chapter is derived in part from the following published and unpunished articles:

– Published as Yebo Feng, Jun Li, and Thanh Nguyen. “Towards Intelligent Application-layer DDoS defense with reinforcement learning.” *Transactions on Information Forensics & Security*, 2023. **In preparation**.

– Published as Yebo Feng, Jun Li, and Thanh Nguyen. “Application-layer DDoS defense with reinforcement learning.” *In 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), pp. 1-10*, 2020 [154].

– Published as Yebo Feng. “Towards Intelligent Defense against Application-Layer DDoS with Reinforcement Learning.” *Computer and Information Science, University of Oregon, Technical Report, DRP-201912-Feng*, 2019 [146].

I am the leading author of the above articles. Most content of this chapter was written by me, and I was responsible for conducting all the system designs, implementations, evaluations, and the presented analyses.

6.1 Introduction

Application-layer distributed denial of service attacks [310], or layer 7 (L7) DDoS attacks, represent a type of malicious behavior that attack the application layer in the network model. These L7 DDoS attacks exploit application-layer messages (e.g.web requests) to swamp specific application functions or components of a victim server (e.g.a web server) to disable or degrade their services, impacting legitimate users’ experience.

L7 DDoS attacks are on the rise and becoming conspicuous threats on today’s Internet. One of the best-known L7 DDoS attacks happened in March 2015 when a massive number of HTTP requests poured towards GitHub [51], causing much reduced availability and higher latency to GitHub's service. This attack worked by injecting nefarious JavaScript code pieces into numerous web pages to redirect a high volume of users' HTTP traffic to GitHub. More recently, Imperva reported a notable L7 DDoS attack [347] in July 2019. This attack was the longest and largest that Imperva has ever seen, lasting 13 days and reaching a peak volume of 292,000 requests per second.

Unfortunately, the detection and defense of L7 DDoS attacks are still not well-studied [104, 310]. Worse, attackers continuously evolve their toolkits and develop more sophisticated L7 DDoS attack techniques. It is therefore compelling to accurately identify L7 DDoS attacks and generate effective mitigation tactics against them.

The key to addressing L7 DDoS attacks is to distinguish L7 DDoS traffic from the legitimate application-layer traffic. This task is difficult, however, given that an L7 DDoS attacker can purposely fabricate application-layer messages that look legitimate, as discussed above. An L7 DDoS message can even be identical to a legitimate application-layer message.

Interestingly, the legitimacy of an application-layer message is heavily dependent on its environment or context. The same application-layer message may be legitimate in one environment, but totally malicious in another. Or similarly, depending on how a client has been interacting with a server in the past, a newly received request from the client may be legitimate in one case, but illegitimate in another. For example, an HTTP GET message is totally legitimate during the routine operation of an HTTP server but could be malicious during an HTTP flooding attack. In another words, it is a Markov decision process to determine whether an application-layer message is legitimate or not.

We thus seek to discover what methodologies would be the most effective in distinguishing L7 DDoS traffic from the legitimate application-layer traffic by considering environmental and contextual factors, instead of only inspecting the messages themselves. This paper proposes the first *reinforcement-learning-based* method that incorporates environmental and contextual factors to distinguish L7 DDoS traffic from the legitimate application-layer traffic. It monitors and analyzes a variety of environmental and contextual factors including those related to the system and network load of the victim server (e.g.disk I/O , CPU operation, memory usage, or link utilization) and the dynamic application-layer behaviors of clients (e.g.request type, size, frequency, and content).

Furthermore, this method streamlines the L7 DDoS defense by integrating the operations of attack detection, message classification, and attack mitigation. Rather than producing labels of each application-layer message for a separate L7 DDoS mitigation module to handle the message, in order to mitigate L7 DDoS attacks, this method directly outputs the action to take for each application message under different circumstances. Actions can include blocking the message upstream, blocking it locally, or postponing its processing.

In addition, this method receives feedback from the actions taken, allowing it to fine-tune what actions are the best for a given situation. With the design of a new multi-objective reward function, this method can determine the most suitable actions to take in a way that (1) minimizes the amount of discarded legitimate messages to provide the service as much as possible to clients when the victim load is low and (2) maximizes the amount of filtered L7 DDoS messages to prevent the server from collapse when the victim load is high.

The evaluation shows that this approach can identify the majority of DDoS traffic and significantly increase the capacity of the victim server. At the peak of L7 DDoS attacks its accuracy is 0.9553 and true positive rate is 0.9873, while when the attacks are not overwhelming the collateral damage is as low as 0. The implementation of this method, while not intricate, provides satisfying performance when running on the server node. With less than 30,000 training episodes, this method can easily adapt to an unacquainted victim server environment.

The structure of this chapter is organized as follows: Related work is described in Section 6.2, followed by an introduction to the threat model in Section 6.3. Subsequently, we elaborate on this approach in Section 6.4 and present an evaluation of this approach in Section 6.5. Open issues, potential future work, and generalizations of the methodology are discussed in Section 6.6. The conclusions drawn from this chapter are encapsulated in Section 6.7.

6.2 Related Work

The current defense models against L7 DDoS primarily follow a two-phase procedure, which performs *detection & classification* to identify the malicious sources or application messages and then *mitigates* the attack by conducting access control. These models treat the two phases as two separate modules, making it difficult to modulate mitigation strategies according to the conditions of specific attacks. On the contrary, our approach considers attack classification and mitigation as an integral whole to pursue the best L7 DDoS defense efficacy. Below we detail the related work in each phase.

6.2.1 Detection & Classification Approaches. We categorize the previous approaches to L7 DDoS detection & classification into three types: *statistical methods*, *learning-based methods*, and *Markov-based methods*.

6.2.1.1 Statistical methods. Researchers build statistical models on both benign and malicious L7 DDoS traffic and then apply them to detect and classify L7 DDoS attacks. For example, DDoS Shield [314] characterizes L7 DDoS attacks on the basis of the application workload parameters that they exploit. It presets the threshold values on the workload parameters according to the measurements and labels the behaviors that exceed the thresholds as malicious; Yatagai et al. [432] proposed a method that detects HTTP GET flood by modeling the browsing order of webpages and the correlation between browsing time and page information size. In general, statistical methods are efficient and steerable. They have decent accuracies in discovering simple L7 DDoS attacks such as HTTP flood and low-and-slow DDoS attack, however, they may have non-ideal effects on handling unseen and complicated attacks.

6.2.1.2 Learning-based methods. As machine learning algorithms are becoming more and more sophisticated, many researchers harness such techniques on big data for detecting L7 DDoS attacks. Seufert et al. [339] proposed a three-layer feed-forward neural network to detect L7 DDoS attacks, using features extracted from the header fields of packets; Yadav et al. [427] applies Stacked AutoEncoder, a deep learning architecture that aims to receive high level features, to generate features from web server logs and build a logistic regression classifier to identify L7 DDoS attacks. Besides, researchers also proposed unsupervised-learning-based detection methods that can extract knowledge or patterns from unlabeled data. ARTP [309] detects L7 DDoS by leveraging the K-means algorithm and performing analysis on features such as request interval, request chain context, and request length. In summary, while a trend, leveraging machine learning in identifying L7 DDoS has mixed results based on the feature extraction method, the system design, and the learning algorithm.

6.2.1.3 Markov-based methods. A Markov model is a stochastic model used to model randomly changing events in probability theory [167]. It assumes that the future state depends only on the current state, and we can infer the next state by performing probability analysis on its past. Works such as [421, 434, 420, 422] track the related behavior of the users and utilize hidden semi-Markov model along with random walk graph to trace the attacks. We consider Markov-based methods as the state of the art because L7 DDoS attacks are *stateful*. As solutions to tackle stateful problems, Markov-based methods can provide fine-grained detection and classification results with decent accuracies. Inspired by this idea, we adopt reinforcement learning to the L7 DDoS problem, which is a Markov decision process that inherits advantages from both learning-based approaches and Markov models.

6.2.2 Mitigation Approaches. As for the mitigation approaches of L7 DDoS, there are mainly two research directions [310]. One is to mitigate attacks on the victim side, such as blocking automated application requests by utilizing user puzzles (e.g., [439, 350]) and setting up specific IPTables or IDS rules [280]. They can reach message-level mitigation granularities but may sacrifice the efficiency of mitigation, since the victim is still required to receive the malicious packets before mitigation, making the system still vulnerable to volumetric L7 DDoS attacks. Another direction is to mitigate L7 DDoS attacks in the network. Once the victim determines the attack sources, it can leverage some traffic filtering or rerouting systems (e.g. [60, 310, 352]) to mitigate attacks from within the network, without consuming any resources on the victim's side. However, they may

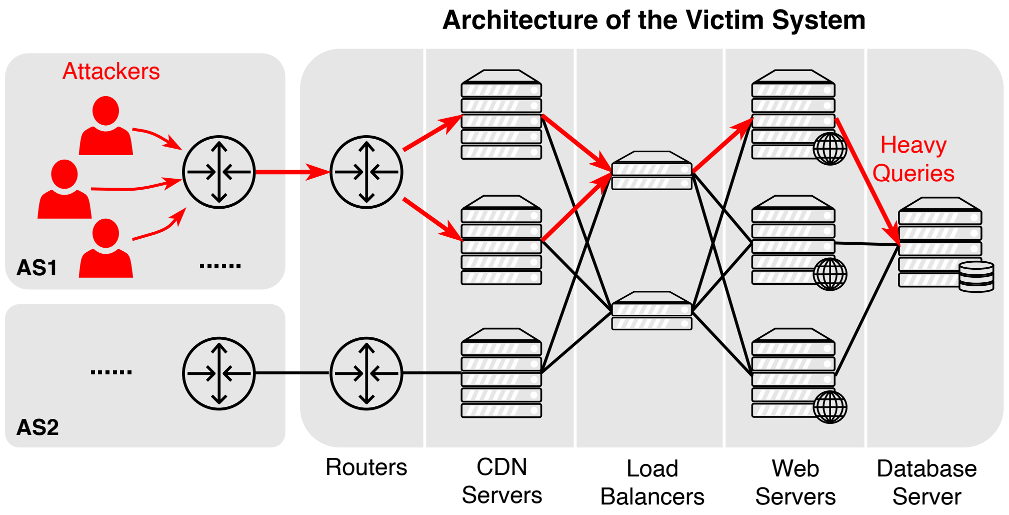


Figure 63. An example of the victim model. The attacker is performing lethal attacks towards the database server of the victim system.

cause a considerable amount of collateral damage since traffic from benign IPs may also be filtered. Our approach, different from the above directions, incorporates both victim-side mitigations and in-network mitigations for efficient and effective defense against L7 DDoS attacks.

6.3 Threat Model

An L7-DDoS victim server can be a single-node application server, or contain many components as illustrated in Figure 63. We assume that L7-DDoS attackers can form a massive botnet to exploit the vulnerability of the victim system, with the source IP addresses of the bots distributed over different autonomous systems (ASes). Also, we assume that the attackers can systematically measure the victim server's operation conditions in order to figure out the vulnerable spot, thus adjusting their attack tactics accordingly.

After investigating the operational models of current L7-DDoS attacks, we categorize L7 DDoS attacks into three types: request flooding attack, leveraged attack, and lethal attack.

6.3.0.1 Request Flooding Attack. In this attack, the attacker overwhelms the system by sending application-layer requests at a high rate from different IP addresses. The attacker's bots may locate in certain IP blocks or distribute all over the Internet among different ASes to make it challenging to identify the attack sources. Then, the botmaster can control the bots to generate requests of any arbitrary frequencies and content to overwhelm the victim.

6.3.0.2 Leveraged Attack. This attack leverages the flaws of the victim system to amplify the threat. Thus it can take down the application server with minimal bandwidth and very few requests. For example, low and slow attacks. The attacker controls bots to utilize tools like R.U.D.Y. [285] or Slowloris [331] to slowly send out the requests to the victim. This procedure keeps many connections to the target server open and holds them open as long as possible, tying up the thread. Other types of leveraged attacks may leverage heavy SQL queries, unbalanced API calls, or flawed message queues to overwhelm the victim with a small amount of application-layer requests.

6.3.0.3 Lethal Attack. In this threat, the attacker first scans the victim system to pinpoint the current performance bottlenecks or vulnerabilities (e.g., I/O, memory space, or database server), which are also called lethiferous spots. Then, the attacker formulates the optimal attack tactics to overwhelm the lethiferous spots. Furthermore, the attacker may adjust attack tactics dynamically based on the condition variations of the victim server to make the attack even more effectual. In general, this intelligent attack is highly threatening to all types of victim systems and difficult to detect due to its dynamics.

6.4 System Design

6.4.1 Overview. In this paper, we assume that L7 DDoS attacks cannot be easily identified through flow-level data since malicious messages will disguise their traffic flows as legitimate. Hence, our solution is on victim-side and considers many factors, such as clients' behavioral information, the network load of the victim server, and the system load of the victim server. We also assume that launching an L7 DDoS attack is a stateful process, just as the process of establishing a TCP connection and collectively sending out the HTTP requests. Thus, we use reinforcement learning (RL) [223], a stateful machine learning technique based on Markov decision process, to construct the attack classification model and formulate appropriate tactics to protect the victim.

RL is a burgeoning area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notions of cumulative rewards. Once the RL agent has made a decision, it gets a reward value to sense whether the current move is suitable or not. Then, it revises the policy to adopt the feedback dynamically. Compared with other L7 DDoS defense approaches, the following advantages make RL more competent to deal with L7 DDoS attacks:

- As a Markov decision process [207], RL aims to maximize the cumulative rewards throughout the monitoring process, which takes advantage of contextual information to infer potential threats.
- In L7 DDoS attacks, the boundary between benign and malicious messages is blurry. Instead of primitively classifying the message as either benign or malicious, the RL agent focuses on formulating appropriate defense tactics that suit current environmental conditions.

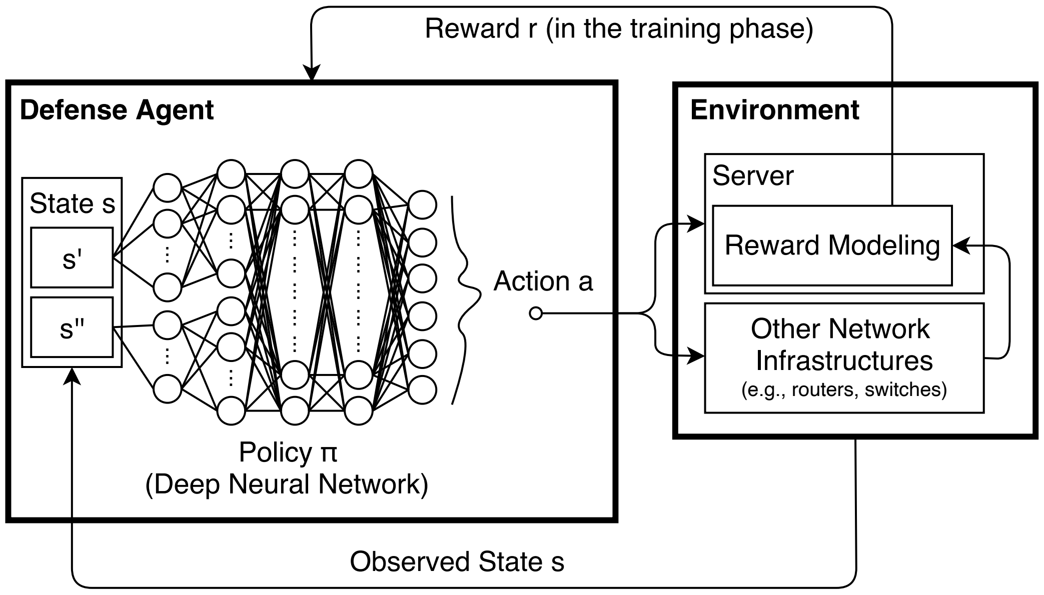


Figure 64. System Architecture.

– RL allows us to use a multi-objective reward function to mentor the agent on constructing an adaptive and dynamic defense strategy against L7 DDoS attacks.

A typical RL system has five elements: *agent*, *environment*, *reward*, *state*, and *action*. The environment is typically stated in the form of a Markov decision process (MDP) and the MDP transition function gives a new state for each incoming application message, processed in sequence. The agent gets the state from the environment (the environment includes the victim server and some related network infrastructures in our case), then sends the next action to the environment. The environment will conduct the action and give feedback to the agent about the suitability of the action by sending a reward value.

Figure 64 shows the detailed system architecture of our approach. The goal is to train a defense policy π in the training phase and apply it in the monitoring phase to defend against L7 DDoS attacks. The victim can be a single node webserver or a complicated server cluster discussed in Section 6.3. If the victim is a large server cluster, the components (e.g., load balancers, databases, and webservers) need to gather their system information to form an aggregated state s and forward it to the defense agent. In the training phase, there is a reward modeling component on the victim-side, which gets access to the ground truth of the simulated traffic. Therefore, the victim server can evaluate the efficacy of the mitigation action and generate the reward value r according to the reward function in real-time. However, once we complete the training phase and put the defense agent into the monitoring phase, the reward modeling component, as well as the reward values, are no longer needed. In the monitoring phase, the defense agent only needs to generate action a according to the observed state s and trained policy π.

The actions generated by the agent have two categories: the victim-side mitigation actions, which only need to be conducted on the server-side (e.g., scheduling actions), and the in-network mitigation actions, which need to be conducted on some external network infrastructures for filtering out specific traffic.

The rest of this section elaborates on the design details of our approach.

6.4.2 States.

The state is represented by a state vector s. In our implementation, each state s has twelve dimensions. Each dimension of s is a value that represents a feature. In this L7 DDoS detection system, we expect s to comprehensively represent both the environmental situations and the current application message's features. Thus, we further divide s into two parts, message state s' and environmental state s'' ($s = s' \cup s''$). State s' summarizes the content of the incoming message and the sender's historical behaviors. It helps the defense agent to infer how abnormal the message or the client is. Meanwhile, state s''

extracts the information from the victim server's system situation. It gives the defense agent a perspective of the whole system's healthy degree.

To calculate the state vector in real-time, we define time t as the primary resolution value. For example, if we want to know the average behavior interval of a client, we only need to sample all its past behaviors during the last time t to calculate the value.

6.4.2.1 Message state s'.

s' is an eight-dimensional vector that extracts eight features from the current application message. It is designed to reflect the historical activities, resource consumption, and behavioral characteristics of the message. The eight features are shown below:

- *$bytes_m$: the number of bytes in a message.*

- *$bytes_b$: traffic size from the message's IP block.* The victim will predefine some IP blocks to classify clients' source IP addresses. $bytes_b$ is a numeric value that indicates the total number of bytes from the incoming message's IP block within time t. This feature is useful to identify request flooding attack.

- *ave: the average behavior interval of the client.* Assume that the client has sent n messages during the last time t, and each interval is denoted by x_i (where $i = 1, 2, ..., n-1$). ave is defined as: $ave = \frac{1}{n-1}\sum_{i=1}^{n-1} x_i$.

- *dev: the average absolute deviation of the client's behavior intervals.* This feature is defined as: $dev = \frac{1}{n-1}\sum_{i=1}^{n-1} |x_i - ave|$.

- *num_m: the number of messages from the client.* This feature is the number of the message sent by this client during the last time t.

- *num_{sm}: the number of all the similar received messages.* For each received message, the server will calculate the number of similar messages within time t promptly. This value plays a crucial role in identifying request flooding attacks,

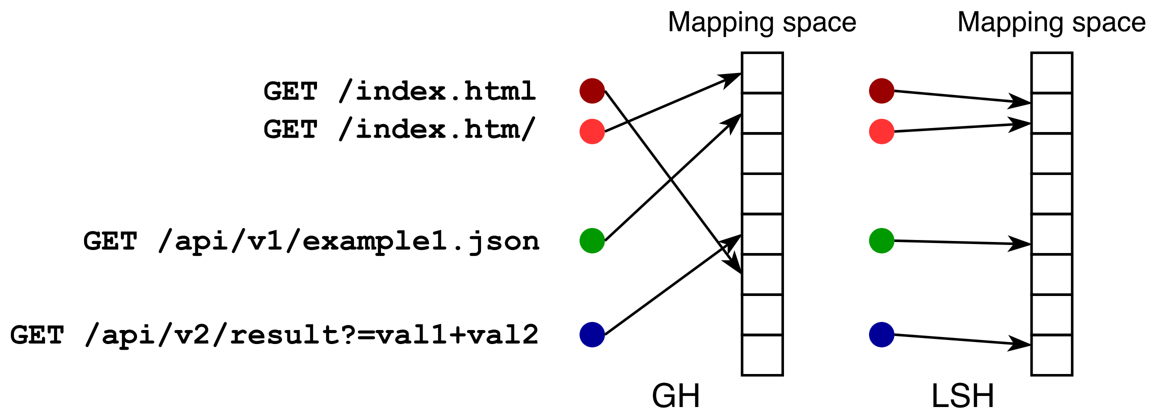


Figure 65. Examples for the general hashing (GH) and Locality Sensitive Hashing (LSH).

leveraged attacks, and lethal attacks. However, calculating this value is expensive, as we need to buffer a considerable amount of messages in the memory and perform complicated string matchings. Thus, we leverage Locality Sensitive Hashing (LSH) [119] to optimize the calculating process.

Different from traditional hashing functions, LSH can output close or identical values from similar input strings, making it efficient in the duplicate checking. Figure 65 shows an LSH example, where the horizontal positions of the four dots represent the difference in their contents. This method requires training before conducting queries, so we collect request message strings that can represent all the application messages that the server can handle, then preprocess the message strings to make them simplified but still informative enough to outline the messages' intentions, behavioral patterns, and the clients' platforms. For example, the message below is a typical HTTP GET message:

```
GET /index.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/4.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.4)
Accept: text/html,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=n465xmdh435may4ib0skrjq360
```

The preprocessing procedure eliminates redundancies in the strings (strings with red color). We then concat the rest of the information in a fixed order, and joint them by deleting all the spaces and line breaks.

```
/index.htmlMozilla/4.0(Windows;U;WindowsNT6.0;en-US;rv:1.9.1.4)text/html,
application/xml;q=0.9,*/*;q=0.8en-us,en;q=0.5ISO-8859-1,utf-8;q=0.7,*;q=0.7300
keep-alivePHPSESSID=n465xmdh435may4ib0skrjq360
```

The original message string turns out to be the string above after the preprocessing procedure, and we use such data to train the LSH function for queries. Whenever there is an input message string m, LSH will input the preprocessed string and generate an output hashing value h. The system will store this hashing value h in a set H with an expiration time of t. Every time the system checks the set H, it will remove all the expired values. We also defined a difference threshold Δ to find similar strings. Therefore, the number of similar messages num_{sm} is defined as:

$$num_{sm} = |\{k | k \in H \wedge |k - m.h| \leq \Delta\}|.$$

• *cons: request consumption.* The sever estimates the consumptions of all the requests that it can handle in advance and builds a precalculated consumption score table. Given a message m, the server will extract the request from m and generate the *cons* value according to the consumption score table. This feature is important to identify leveraged attacks.

• *φ: the ratio of incoming traffic size to outgoing traffic size.* The server estimates the outgoing traffic size $bytes_o$ if it responses this message, then calculate the ratio by: $\varphi = bytes_m / bytes_o$.

6.4.2.2 Environmental state s''. s'' is a four-dimensional vector that extracts four features from the server and network's current conditions. This vector is supposed to be a good representative of the environmental metrics so that the agent can correctly infer how dangerous the server's condition is and what is the system bottleneck currently. The four features are shown below:

• *$util_{cpu}$: CPU utilization.* This value is the occupancy rate of the CPU. If the victim system has multiple servers, $util_{cpu}$ is equal to the maximum CPU occupancy rate in the cluster.

• *$util_{mem}$: memory utilization.* This value is the occupancy rate of the memory. If the victim system has multiple servers, $util_{mem}$ is equal to the maximum memory occupancy rate in the cluster.

• *$util_{link}$: link utilization.* This value is the occupancy rate of the link bandwidth. If the victim system has multiple link, $util_{link}$ is equal to the maximum link occupancy rate in the system.

• *$eutil_{link}$: expected link utilization.* If the victim has statistical data about the expected link utilization rates in different periods of the week, this value is the expected link utilization rate in an ordinary situation. Otherwise, this value is the $util_{link}$ during the previous time t.

6.4.3 Actions. As discussed in Section 6.2, individuals can utilize in-network and victim-side mitigation approaches to defend against L7 DDoS attacks. We further derived six types of particular actions (shown as below) that an agent can take in the defense process. Each of the action a targets some specific circumstances.

- Action a_i: enabling the server to receive and respond the current application message ordinarily.
- Action a_{ii}: enabling the server to receive the current message ordinarily but postpone the processing procedure.
- Action a_{iii}: drop the current application message on the victim-side.
- Action a_{iv}: drop all the application messages that have the content similar to the current message on the victim-side.
- Action a_v: blocking all the traffic from the IP address of the current application message in the upstream router.
- Action a_{vi}: blocking all the traffic from the IP block of the current application message in the upstream router.

a_i and a_{ii} are *scheduling actions*, a_{iii}, a_{iv}, a_v, and a_{vi} are *defensive actions*. In another taxonomy, a_i, a_{ii}, and a_{iii} are *single-targeted actions*, which only affect the current application message. a_{iv}, a_v, and a_{vi} are *multiple-targeted actions*, which affect a group of application messages.

Conducting defensive actions on particular messages does not necessarily mean the messages are malicious because the agent can choose to sacrifice some false positive rates to ensure the functioning of the server during a severe attack. Similarly, conducting scheduling actions on a particular message does not guarantee the legitimacy of the message. If the system is on idle time, and the malicious message cannot cause some real harm against the server, the agent will take conservative strategies to minimize collateral damages.

6.4.4 Reward Function.

The overall objective of the reward function is mentoring the defense agent to form a defense policy to fulfill the following requirements in the training phase:

- When system occupation rate is low, minimize the false positive rate of mitigation to ensure all the legitimate messages can be properly processed.
- When system occupation rate is high, maximize the true positive rate of mitigation to block all possible attacks in order to prevent the system from crushing.
- The agent is encouraged to conduct multiple-targeted actions rather than single-targeted actions so that the agent can discover rules in the attacks instead of inefficiently labeling every single message.

In order to address the goals above, we construct a piecewise function $R(a(m))$ as the reward function to mentor the agent conduct suitable actions on correct messages. Here, $a(m)$ denotes conducting action a on message m. In the training phase, whenever the action a is placed, the reward modeling component will use $R(a(m))$ to calculate the reward value r, telling the defense agent how suitable the current action a is. In this paper, if the agent conducts defensive actions on the legitimate messages, we consider these messages as false positive samples, and vice versa. We also define γ the system occupation rate, which is calculated as:

$$\gamma = max(util_{cpu}, util_{mem}, util_{link}).$$

Additionally, $|fp|$ denotes the number of false positive samples, $|tp|$ denotes the number of true positive samples, $|fn|$ denotes the number of false negative samples, and $|tn|$ denotes the number of true negative samples. We define a policy transition threshold value α to decide when the agent should adjust the defense policy to minimize the false positive rate or to maximize the true positive rate. In this paper, we set α as 0.75.

When $\gamma < \alpha$, we set the reward function $R_1(a(m))$ for single-targeted actions as Equation 6.1.

$$R_1(a(m)) = \begin{cases} -2 & for\ false\ positive\ sample \\ 1 & for\ true\ positive\ sample \\ -1 & for\ false\ negative\ sample \\ 0 & for\ true\ negative\ sample \end{cases} \tag{6.1}$$

This reward function gives the agent more penalties when false positive generated, which aims to constraint the agent to ensure all the possible legitimate messages can be properly processed when the system load is within a safe zone.

In the scenario that the agent is making multiple-targeted actions, and $\gamma < \alpha$. Assume that the action a will affect a set of messages $M = \{m_1, m_2, ..., m_n\}$, we set the reward function $R_2(a)$ as Equation 6.2.

$$\begin{aligned} R_2(a(M)) &= \eta \sum_{i=1}^{n} R_1(a(m_i)) \\ &= \eta(-2|fp| + |tp| - |fn|) \end{aligned} \tag{6.2}$$

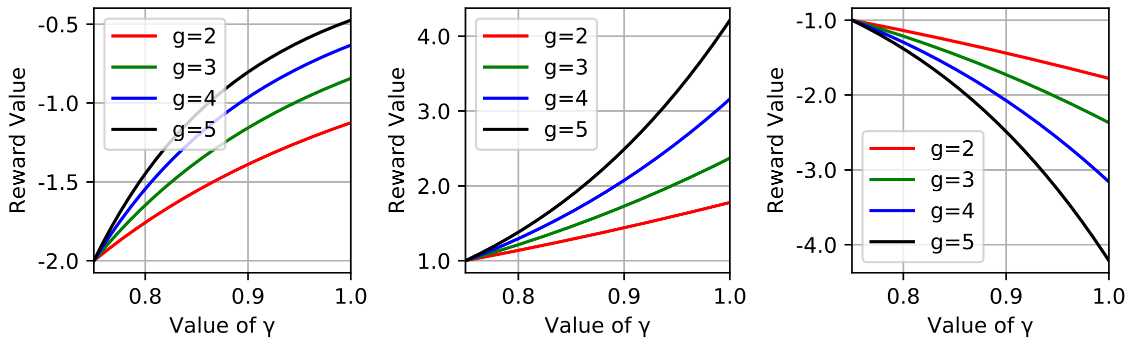
Where η is the reward multiples. We can set η as a value more one so that the agent would get extra rewards or penalties when making multiple-targeted actions. The larger η is, the more the agent is encouraged by the reward functions to take multiple-targeted actions for conducting the defense policy effectively. This mechanism is necessary for the defense agent because monitoring a large amount of incoming messages is an expensive operation and could become the a system vulnerability itself. The agent can fix this problem by frequently generating multiple-targeted actions.

When $\gamma \geq \alpha$, the victim system is heavily loaded, which means the highest priority of agent is to mitigate as many L7 DDoS attacks as possible to guarantee the proper functioning of the server. In this scenario, we set the reward function $R_3(a(m))$ for single-targeted actions as Equation 6.3.

$$R_3(a(m)) = \begin{cases} -\frac{2}{(\frac{\gamma}{\alpha})^g} & for\ false\ positive\ sample \\ (\frac{\gamma}{\alpha})^g & for\ true\ positive\ sample \\ -(\frac{\gamma}{\alpha})^g & for\ false\ negative\ sample \\ 0 & for\ true\ negative\ sample. \end{cases} \tag{6.3}$$

Where g is the hazard index, an input parameter that determines how eager the victim wants the attack to be mitigated. The larger g is, the more tactics shifts the agent will have according to the environment, but g should always be larger or equal to 1. Figure 66 shows the curves of the reward function in this scenario with different g values (we set $\alpha = 0.75$ in the curves), we can intuitively see the variation of the reward functions based on the change of γ. The agent will get less and less penalties from false positive samples with the increasing of γ. Conversely, both the rewards from true positive samples and the penalties from false negative samples will rise significantly. This reward function design will constraint the agent to identify and block as many malicious application messages as possible, with the cost of sacrificing a little bit false positive rate.

In the scenario that the agent is making multiple-targeted action a on a set of message M ($M = \{m_1, m_2, ..., m_n\}$), and $\gamma \geq \alpha$, we set the reward function



(a) For false positive samples.

(b) For true positive samples.

(c) For false negative samples.

Figure 66. Single-targeted Reward Functions for $\gamma \geq \alpha$ and $\alpha = 0.75$.

$R_4(a(M))$ as Equation 6.4:

$$\begin{aligned} R_4(a(M)) &= \eta \sum_{i=1}^{n} R_3(a(m_i)) \\ &= \eta(-\frac{2}{(\frac{\gamma}{\alpha})^g}|fp| + (\frac{\gamma}{\alpha})^g|tp| - (\frac{\gamma}{\alpha})^g|fn|) \end{aligned} \tag{6.4}$$

$R_4(a(M))$ is in direct proportion to the summation of reward values that returned by all the affected messages. Still, we use the reward multiples parameter η to encourage the agent to take multiple-targeted actions rather than single-targeted actions.

6.4.5 Training. The training of the deep reinforcement learning agent follows Q-value iteration [410]. For every state s, the agent will generate an action a, which creates a state-action pair. The reward function will also return a reward value r based on the state-action pair, therefore, we define a function Q that calculates the quality of a state-action combination: $Q : s \times a \rightarrow r$.

At time i, assuming the agent is located in s''_i and receives a message state s'_i, it will select an action a_i to take. After the agent observed the reward r_i, it will enter a new environmental state s''_{i+1} and update the value of Q. The core of the algorithm is the value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_i, a_i) \leftarrow (1-\beta)\cdot Q(s_i, a_i) + \beta \cdot (r_i + \zeta \cdot \max_a Q(s_{i+1}, a)),$$

where β is the learning rate, and ζ is the discount factor.

However, the state s we use in this schema is a twelve-dimensional vector, which could generate too large value space for the system to cover in both training and monitoring phases. To tackle this problem, we use a deep neural network to serve as a likelihood function for estimating the $Q(s, a)$.

Just as the topology diagram in figure 64 shows, we leverage a five-layer neural network to approximate the policy function. There are three hidden layers, one input layer, and one output layer in the neural network. The input layer has 12 nodes to import the state vector s, and the output layer has six nodes to generate the recommendation rates for six possible actions respectively. The second and fourth layers have 14 nodes, while the third layer has 15 nodes. A unique aspect about neural network is that the first and second layer are not fully connected. Instead, we separate the nodes for s' from s'' to ensure that the neural network can treat the two sub-state vectors differently.

The training of the agent is similar to the training of ordinary neural networks, in which we define a loss function to measure how good the agent's tactic is. The loss is a value that indicates how far our action a is from the actual target:

$$loss = (r + \epsilon \max_{\acute{a}} \hat{Q}(s, \acute{a}) - Q(s, a)),$$

where ϵ is the decay rate, and $r + \epsilon \max_{\acute{a}} \hat{Q}(s, \acute{a})$ is the actual target. The training of the neural network is also the process of minimizing the *loss* value with back propagation.

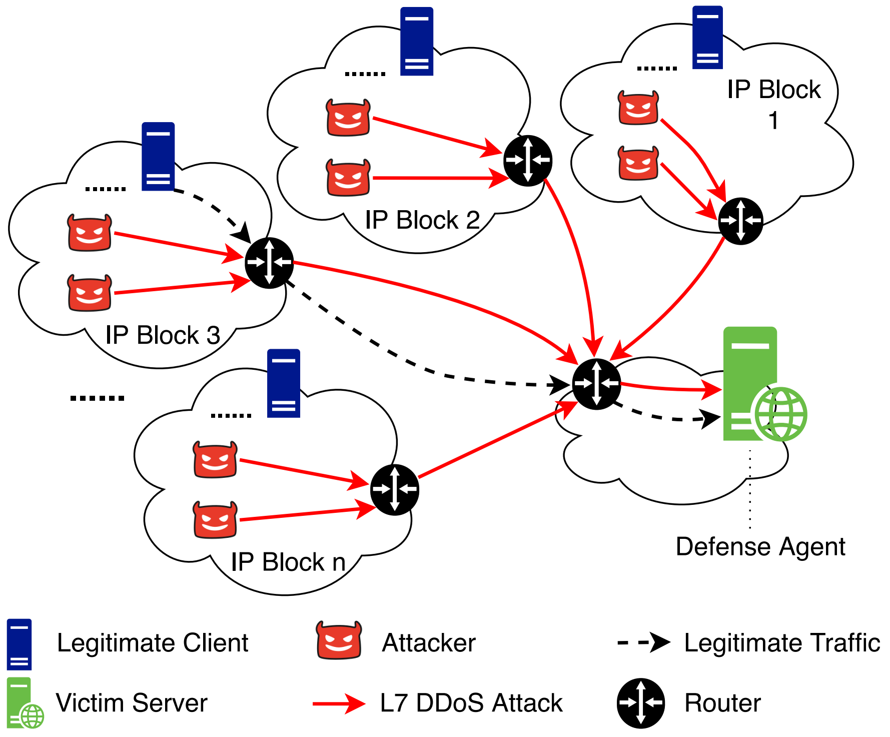


Figure 67. Topology of Simulated Network Environment.

Each state s consists of a message state and an environmental state ($s = s' \cup s''$). The agent will continuously get environmental state s'' but only get message state s' when there is an incoming application message. Thus, the agent will only be activated when it receives s' in both training and monitoring phases.

6.5 Evaluation

6.5.1 Implementation and Simulations. We utilized Open vSwitch [306] and Mininet [237] to construct the simulation environment. Figure 67 shows the basic topology of the simulated network environment. There are n IP blocks in this network; each of them has 5 legitimate clients and 5 malicious clients. Besides, we constructed the RL-based L7 DDoS attack defense system with OpenAI Gym [77] and Keras [98].

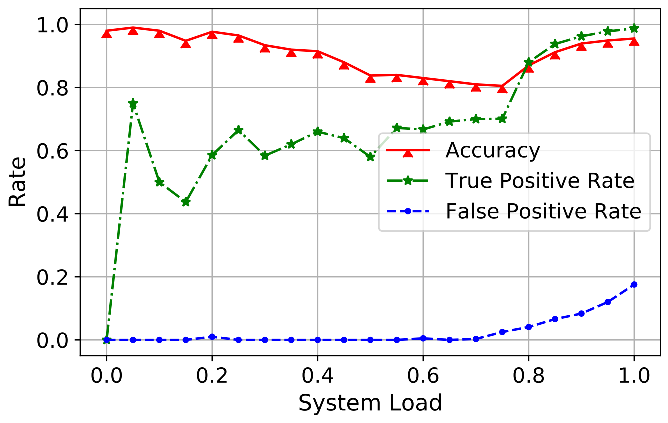


Figure 68. Performance Metrics of Different System Loads.

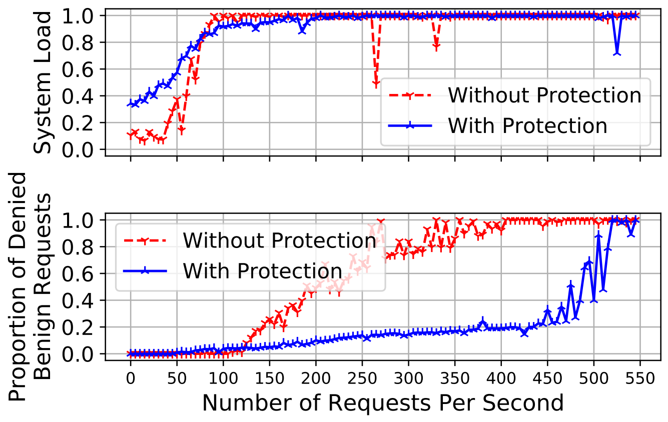


Figure 69. Efficacy of Attack Mitigation.

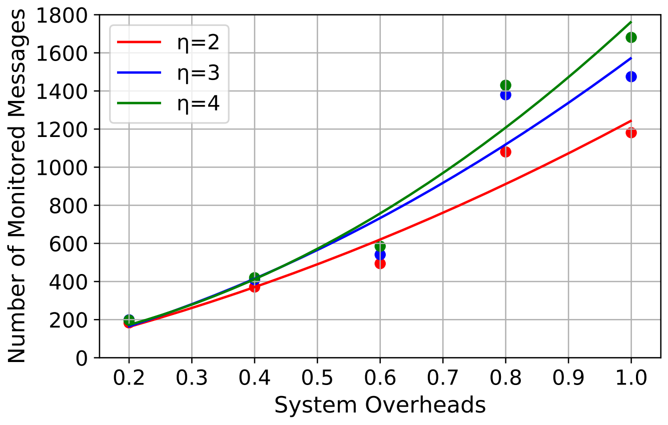


Figure 70. System Overheads for the Defense Agent.

We simulated a victim system by constructing a Node.js web server that handles HTTP requests and SMTP requests. The server runs on a virtual machine with 6GB RAM and a 4-core 2.0 GHz CPU. It also maintains an HTTP-based API that can read its hard disk and return selected images. The API is a designed performance bottleneck (lethiferous spot) for attackers to exploit.

For L7 DDoS attacks, we used simulated traffic rather than captured traffic because L7 DDoS attacks are diverse — malicious messages in one environment can be legitimate in another. Moreover, there are few packet-level L7 DDoS traffic available in public repositories. The majority of the existing public L7 DDoS datasets are log files or preprocessed features. Thus, we used the Application Layer DDoS Simulator [357] to simulate request-flooding attacks. For leveraged attacks, we used Slowloris [331] to simulate the most typical leveraged attack — low and slow attack. In the end, we used modified HULK program [346] to generate lethal attacks towards the known performance bottleneck.

Based on our empirical studies (of which we skip the details for space considerations), we set some of the parameters in this approach as follows: for the number of IP blocks n in the evaluation, we set it to be 10; for the policy transition threshold value α, we set it to be 0.75; the learning rate β for agent training is 0.25 in this implementation; for the hazard index g, we set it to be 3.

6.5.2 Ability of Mitigating Attacks. Although the proposed method does not need to generate the precise labels of incoming messages, we can still evaluate its ability to mitigating L7 DDoS attacks by inferring the correctness of output action a. As indicated in Section 6.4.3, we count messages that trigger scheduling actions as legitimate requests. Conversely, we count messages that trigger defensive actions as malicious messages. All the evaluation metrics in this section are based on this regulation.

We simulated benign messages and launched the L7 DDoS attacks to the victim simultaneously for evaluating the agent's accuracy of mitigating L7 DDoS attacks. During the test, we firstly ensured the volume of legitimate messages was always under the victim server's capacity so that the server would not crash due to legitimate activities. Afterward, we adjusted the amount of malicious messages to test the performance of this approach with different system loads. Here, we consider the system load as the system occupation rate γ defined in Section 6.4.4.

Figure 68 shows the trends of mitigation accuracies, false positive rates, and true positive rates during different system loads (we consider malicious messages as positive samples in this paper), where the y-axis represents the system workload, and the x-axis represents the rate value.

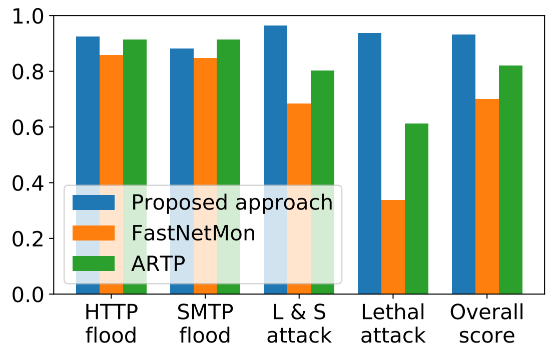
When the system load is at a low rate, we can get a nearly 100% mitigation accuracy, since the majority of the messages are benign, the agent will minimize the false positive rate at this point. However, when both the system workload and the volume of attacks are increasing, the accuracy has some apparent drops. Although the attack volume increased, the defense agent still uses the defense tactic that aims at minimizing the false positive, guiding the agent to sacrifice the true positive rate for letting the server adequately process most of the legitimate requests. Thus, the false positive rate remains approximately zero within this zone. On the contrary, the trend for true positive rates fluctuates in low system-load scenarios, because the volume of malicious requests is still low, making it hard to reach statistical significance.

The transition comes in when the system workload is at 0.75. From this point, the defense agent assumes that the server system is in hazardous conditions, so it has to mitigate as many attacks as possible to protect the victim server. As the system load goes higher, the value of $(\frac{\gamma}{\alpha})^g$ in the reward functions becomes larger, and the false positive rate becomes less and less critical. Hence, we can distinctly see that the defense agent starts maximizing the true positive rate. This sacrifices some false positive rates but still increases the overall accuracies. In the end, when the system load stabilizes at 100%, the accuracy, true positive rate, and false positive rate are 0.9553, 0.9873, and 0.1756, respectively.

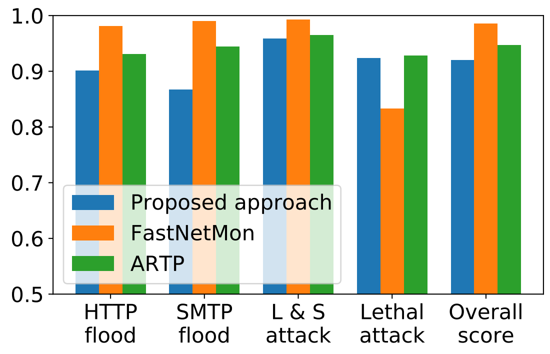
In brief, this evaluation result proofs that the reinforcement learning agent can intelligently formulate applicable tactics to defend against L7 DDoS attacks, and the mitigation accuracies of the tactics are satisfactory.

6.5.3 Mitigation efficacy. We evaluated the efficacy of our approach and presented the results in Figure 69. The x-axis in the figure represents the number of application messages made to the victim server per second, including both the legitimate messages and malicious messages. The y-axis of the upper subplot represents the system load, while the y-axis of the lower subplot represents the proportion of denied benign messages.

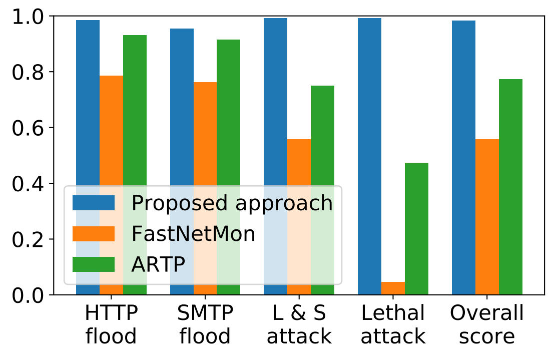
Initially, the resource consumption of the server without protection is lower than the server with the agent running because the deployment of the defense agent costs a certain amount of computing resources, especially for maintaining the LSH function, message monitoring, and the operation of the deep neural network. However, this consumption will pay back shortly with the increasing number of receiving messages. We can see that the proportion of denied benign messages increases observably for the server without protection. If we assume that a server



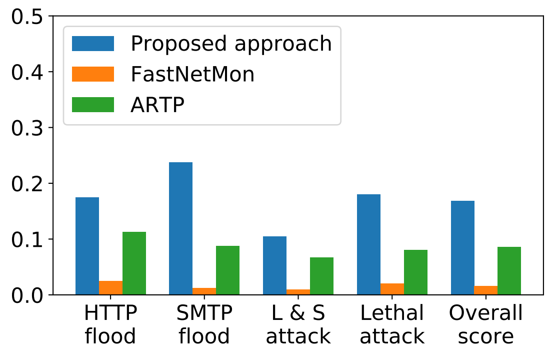
(a) Accuracy.



(b) Precision.



(c) Recall.



(d) False Positive Rate.

Figure 71. Results of Comparison Evaluation.

is considered to be proper functioning when the deny rate of legitimate messages is lower than 20%, the capability of the server without protection is approximately 140 messages per second. After reaching 250 messages per second, the server without protection is almost useless, with the majority of message requests getting denied. While for the server with the defense agent's protection, the deny rate of legitimate messages goes higher than 20% only after the number of messages per second hitting 440, which is **3.15** times the capability of the unprotected server.

Therefore, this approach can significantly enhance the service capability of the server and make the victim resilient during some severe L7 DDoS attacks.

Additionally, running the defense agent requires system overheads. Figure 70 shows the system overheads for the defense agent when monitoring different numbers of messages per second. We experimented by removing the server function of the victim system. Hence, all of the computing resources were devoted to the defense agent, and we can directly measure the system overheads. From previous experiments, we already know that the capability of the server without protection is approximately 140 messages per second. In this figure, we can see the agent can monitor 1.5 times the maximum messages that the server can process with less than 20% of computing resources. Besides, the larger the reward multiples η is, the more the agent is encouraged to take multiple-targeted actions, thus increasing the monitoring efficiency. We set η as 2 in other experiments. However, if we increase the value of η to 4 and devote all the system overheads to the agent, the agent can monitor nearly 1800 messages per second, which is more than 12 times the server's maximum processing capability.

6.5.4 Comparison evaluation. We also compared our approach with two other DDoS attack detection approaches. One is FastNetMon [289], a commercial DDoS detection software that applies statistical methods. Although this software is not designed for L7 DDoS attacks, it offers good performance on general DDoS detection. Another is ARTP [309], a learning-based detection approach particularly designed for L7 DDoS attacks.

To evaluate their performance, we simulated the traffic of HTTP flood, SMTP flood, low and slow attacks, and lethal attacks. Figure 71a shows the accuracies of these approaches. Our approach achieves the best accuracy scores for detecting HTTP flood, low and slow attack, and lethal attack. ARTP only slightly exceeds our approach in detecting SMTP flood. Although our approach does not have perfect scores in precision and false positive rate (as shown in Figure 71b and Figure 71d), it still accomplishes the initial design objective, which is to sacrifice a little bit false positive rate to block as many malicious requests as possible during the peaks of attacks. As we can see from Figure 71c, our method achieves the best recall scores in identifying all types of attacks because it can adjust the mitigation strategies dynamically based on the condition of the victim server.

6.5.5 Robustness in different environments. We have noticed the importance of the robustness of RL-based approaches. Therefore, we deployed the trained agent in different environments to evaluate the adaptability of our approach. Figure 72 shows the evaluation results of robustness. Environment 2 is slightly different from the trained environment, which is assigned with a 4GB RAM and a 3-core 2.0 GHz CPU. Environment 3 has the same hardware as environment 2 but runs different application services, which offers video streaming and download services with HTML5. Environment 4 has a 12GB RAM and a 8-core 2.7 GHz CPU, which is quite a different hardware environment compared with others.

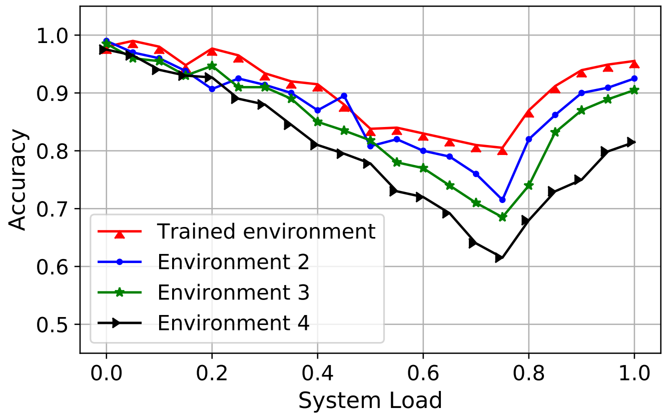


Figure 72. Accuracies in Different Environments.

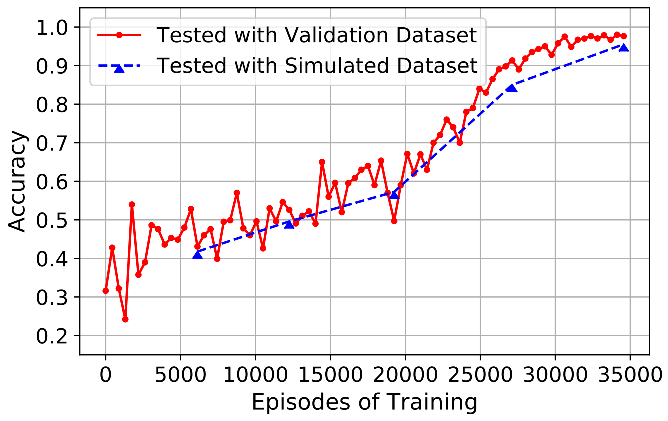


Figure 73. Convergence Trend while Training.

The evaluation results show that the accuracies of our approach only drop a little bit if the environment changed slightly. Even if the application service changed in environment 3, the agent could still achieve around a 90% accuracy at the attack peak. In fact, the design philosophy of our approach promotes adaptability. For instance, many of the features we designed are proportions rather than an absolute value; we avoided using the application-specific features in the state.

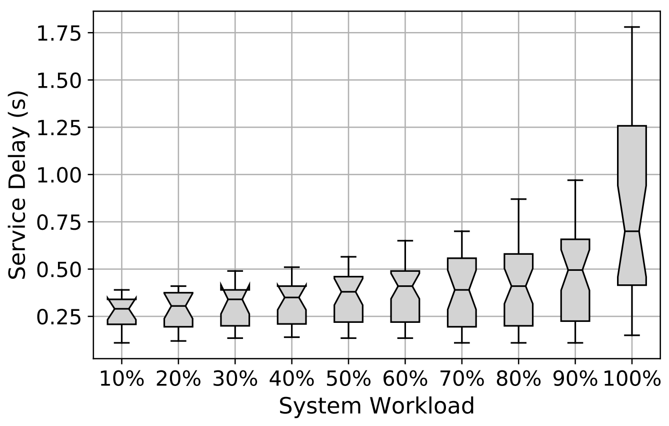


Figure 74. Box Plot for Service Delay.

6.5.6 Agent training. In the training phase, we trained the defense agent in the platform for 80 hours, with nearly 35,000 episodes. We recorded the L7 DDoS attack detection accuracies during different stages to evaluate the efficiency of agent training. As we can see from the results (Figure 73), the training process goes relatively slow and precarious during the first 20,000 episodes. Then it evolves quickly from around 65% accuracy to more than 90% accuracy in the next 7,500 episodes, enabling the defense agent to offer decent protection to the victim server. Eventually, the accuracy of the agent stabilizes near 96% after 30,000 episodes of training. This evaluation also proves that it is feasible to retrain the defense agent within half a week to fit a whole new environment in a real deployment.

6.5.7 Service delay. Since the defense agent will continuously inspect all the incoming application messages during the server operation, the service delay could be an underlying concern that impacts the user experience. Therefore, we measured the lengths of delays under different system workloads and presented the results in a boxplot (as shown in Figure 74).

Here, we define the length of delay as the time duration from sending out a message to receiving the whole reply. A delay less than 0.5 seconds is imperceptible to the users. As we can see in the Figure 74, the average delay time for the server remains under 0.5 seconds when the system workload is less or equal to 90%. Although the delay without any defense approaches implemented is around 0.25 seconds, the presence of the defense agent is still unremarkable to the clients most of the time. Even when the system workload reaches 100% and the attackers are trying to overwhelm the victim server, the service delay can still lay within an acceptable range (0.4 seconds to 1.25 seconds). Meanwhile, the system without any protections is already in an unusable condition under this circumstance.

6.6 Discussions

In this section, we examine the limitations and potential future developments related to our approach, while also considering its prospective suitability for a broad array of other applications.

6.6.1 Limitations and Open Issues.

In most cases, this reinforcement-learning-based approach can accurately identify L7 DDoS attacks and generate defense tactics simultaneously. However, it has some limitations:

1. Application-layer DDoS attack is highly dependent on the victim system and the network environment, so are the attack detection and defense solutions. Thus, this approach may require us to retrain the agent to accommodate the current victim system when the environment is changed dramatically.

2. This method is a learning-based approach, although the training data of this approach can be enhanced to cover more attack models, thus improving the capability of this approach, nonetheless, if the training does not include information from zero-day attacks, this method probably will not be able to deal with them.

This approach also faces several open issues as possible future working items:

1. Currently, we mainly use HTTP flood, SMTP flood, low and slow attack, and API-based lethal attack to train the agent. However, there are still many other types of L7 DDoS attacks existing such as XML-based attack. It would be a meaningful improvement to simulate more types of attacks in the training phase to make the reinforcement learning agent more robust.

2. The performance bottleneck of this approach is in the victim server, because the victim server takes charge of both the message decoding and environmental information transmission. We can further increase the efficiency of the RL agent by using more advanced data structures or algorithms.

3. As a learning-based approach, this method could be vulnerable to adversarial attacks. It is significative to develop some systematic methodologies to evaluate and improve the robustness of this method.

6.6.2 Methodology Generalizations. To defend against L7 DDoS, we have proposed several new algorithms, techniques, and data processing pipelines. These contributions can potentially be generalized to other FGTA applications.

6.6.2.1 Reinforcement learning for attack defense. In our proposed methodology, we have incorporated the concept of reinforcement learning as a mechanism for attack defense. This pioneering approach employs a model that accrues insights from the surrounding environment and historical actions, thereby making more enlightened decisions regarding potential attack defense. The implementation of reinforcement learning in this context serves as an influential instrument as it equips the system with the capacity to adapt to evolving attack patterns and tactics. Its application is not confined to the defense against L7 DDoS attacks; there exists potential for its generalization in defending against other variants of network attacks. Moreover, this reinforcement learning approach can also be extended to other FGTA applications that necessitate dynamic and adaptable response mechanisms.

6.6.2.2 Action-oriented pipeline. The L7 DDoS defense pipeline that we've proposed in this chapter offers a departure from the conventional FGTA pipeline which initially identifies malicious traffic sources before proceeding to block them. Our proposed pipeline directly formulates the most apt defense actions tailored to the current situation, striving to minimize collateral damage when the victim system load is low, and to optimize the true positive rate when the victim system load is heavily burdened. Its focus isn't exclusively on achieving the highest detection accuracy across all scenarios, but rather on ensuring the most effective defense. This pipeline could feasibly be extended to other FGTA applications or defense systems necessitating dynamic and responsive mechanisms.

6.6.2.3 Collaborative defense strategy. In our proposed methodology, we have further integrated the concept of collaborative defense as a strategy for attack mitigation. This approach amalgamates FGTA with endpoint-based defense to guard against L7 DDoS attacks, consequently enhancing application-layer situational awareness and bolstering the efficacy of attack defense. This collaborative defense strategy could be extended to encompass other security applications that necessitate awareness of the environmental context or other domain-specific data. For instance, in the realm of malware detection, there lies potential for researchers to leverage both FGTA and Threat Intelligence Systems (TISs) to augment the defense against malware invasions.

6.7 Conclusions

L7 DDoS attacks are becoming increasingly sophisticated and threatening, outpacing the severity of traditional DDoS attacks. Compared to traditional DDoS attacks, L7 DDoS attacks present a substantial challenge to conventional FGTA methodologies. This difficulty predominantly stems from these traditional approaches lacking the essential application-layer situational awareness that's crucial for the effective detection and mitigation of such advanced attacks.

Consequently, in this chapter, we investigate the joint use of endpoint-based defense and FGTA to effectively combat L7 DDoS attacks. Specifically, we propose a reinforcement-learning-based approach that can self-evolve according to the interactions with the environment. It continuously monitors and analyzes a variety of metrics related to the server's load, the dynamic behaviors of clients, and the network load of the victim, to detect and mitigate L7 DDoS attacks, including choosing the most appropriate mitigation tactic. Different from typical DDoS detection approaches that label the traffic as either legitimate or malicious, this approach employs a new multi-objective reward function that minimizes false positive rate to avoid collateral damage when the victim system load is low and maximizes the true positive rate to prevent the server from collapse when the victim system load is high. Evaluation shows that this approach protects a victim server from L7 DDoS attacks effectively; it can mitigate 98.73% of the malicious application messages when the victim is brought to its knees and achieve minimal collateral damage when the L7 DDoS attack is tolerable.

CHAPTER VII

FUTURE RESEARCH AVENUE

In this chapter, we discuss the broader future research directions in the realm of FGTA for malicious activity detection. Although previous chapters have addressed open issues and potential advancements specific to individual proposed methods in corresponding sections, this chapter takes a more overarching view by concentrating on the overall future research avenues within the FGTA domain.

7.1 Improving the Analysis Coverage and Efficacy

Throughout the years, researchers and developers have introduced a variety of FGTA-based techniques for detecting an extensive array of malicious activities, including DDoS attacks, botnet activities, malware infections, and worm propagation. Despite these advancements, several use cases and scenarios have yet to be addressed by existing FGTA methods. For example, the emergence of UAVs has led to the development of novel, UAV-specific attacks that necessitate detection. The recent SolarWinds supply-chain attacks [416] have underscored the need for FGTA techniques capable of identifying such threats. Furthermore, FGTA can be utilized to oversee and protect network traffic in autonomous vehicles. Consequently, a more extensive detection coverage of FGTA methods is required to tackle the increasing number of use cases and scenarios, encompassing new applications, attacks, and communication protocols. To accomplish this, researchers must first investigate the mechanisms of each novel malicious activity to establish a comprehensive understanding of the threats. This foundational knowledge will enable them to identify the unique characteristics and behaviors of these threats. Subsequently, they can choose to either adapt existing FGTA techniques or create new ones tailored to the detection of these emerging malicious activities. By continuously refining and expanding the scope of FGTA methods, researchers can ensure that these approaches remain effective and relevant in the ever-evolving landscape of cyber threats.

Additionally, as discussed in Chapter II, the effectiveness of numerous current FGTA methods is not optimal for real-world deployment. Based on observation points, FGTA approaches may encounter millions of traffic flows within a brief period in real-world settings. Under these circumstances, an FGTA method could produce a substantial number of false positives or false negatives, even if it achieves over 95% accuracy in closed-world evaluations. As a result, enhancing the efficacy of FGTA remains a perennial subject of interest for researchers and developers. To achieve this, researchers and developers should focus on identifying factors contributing to false positives and negatives, and leverage appropriate and optimized data processing algorithms to tackle these factors. Besides, by integrating domain knowledge and expert insights, more context-aware solutions can be developed to better handle the complexities of real-world environments, thereby improving the overall efficacy of FGTA methods.

7.2 Enhancing the Adaptability to Other Network Environments

As the complexity and diversity of network environments continue to grow, it becomes increasingly vital for FGTA methods to maintain their adaptability across a broad range of scenarios. While in Chapter V we have proposed a DDoS detection method demonstrating strong adaptability, many existing FGTA methods still struggle with this issue. Often, FGTA techniques developed for a specific network environment may not be directly applicable to others due to differences in network topology, traffic patterns, and other factors. To address this challenge, we suggest the following strategies for enhancing the adaptability of FGTA techniques across various network environments.

1. **Model transferring:** Employing model transfer techniques, where pretrained models are fine-tuned for specific network environments, can help improve the adaptability of FGTA methods. By leveraging the knowledge gained from one network setting, researchers can expedite the training process and enhance detection performance in other, potentially related environments. This approach promotes efficiency and adaptability in addressing a wide range of network scenarios.

2. **Online or incremental learning:** Incorporating online or incremental learning approaches [341, 163, 202] allows FGTA methods to adapt to changing network conditions in real-time. These techniques enable continuous model updates as new data becomes available, ensuring that the detection methods remain relevant and effective in the face of evolving network dynamics.

3. **Adaptive feature engineering:** Developing adaptive feature engineering techniques can help FGTA methods better capture the unique characteristics of different network environments. By dynamically selecting and adjusting relevant features regardless of the network environments, these approaches can optimize detection performance across a wide range of scenarios.

4. **Cross-domain learning:** Similar to adaptive feature engineering, leveraging cross-domain learning techniques can enable FGTA methods to easily transfer knowledge and insights from one network environment to another. By exploiting the shared underlying structures and patterns between networks,

researchers can develop more robust and adaptive models that can generalize well across various settings.

5. **Customizability and modularity:** Designing FGTA methods with customizable components and modular architectures allows for seamless integration and adaptation to specific network environments. By creating reusable and interchangeable modules, researchers can more easily tailor the methods to the unique characteristics and requirements of different networks.

7.3 Enhancing the Explainability

Explainability is another critical aspect of FGTA-based detection methods, as it facilitates better understanding, trust, and adoption of these techniques by network administrators or security analysts. However, it is often ignored by most existing FGTA methods. By providing clear and interpretable explanations of how a particular method detects malicious activities, users can make more informed decisions when managing and securing their networks.

One approach to enhancing the explainability of FGTA methods is to leverage interpretable models. Designing inherently interpretable models, such as decision trees and linear models, can offer transparent and easily understandable insights into the decision-making process of FGTA methods. By prioritizing interpretability in model development, researchers can create solutions that are more readily adopted by end-users.

Another direction is to develop post-hoc explainability techniques. Implementing post-hoc explainability techniques, such as LIME [173], SHAP [75], or counterfactual explanations [278], can help elucidate the decision-making process of complex models, such as deep learning and ensemble methods. These techniques provide insights into the importance and influence of specific features on the model's predictions, enabling users to better understand the underlying logic.

Last but not least, researchers can also develop visualization techniques to represent the inner workings of FGTA-based methods, thereby enhancing the explainability. By presenting complex data and model structures in a visually accessible manner, users can more easily grasp the relationships and patterns driving the detection process.

7.4 Expanding and Diversifying Traffic Data Collection for FGTA Developments

High-quality, diverse, and representative traffic data is essential for the development and evaluation of effective FGTA methods. By expanding and diversifying traffic data collection, researchers and developers can ensure that FGTA techniques are robust and adaptable to various network environments and attack scenarios. Nonetheless, there is currently a shortage of high-quality traffic data representing various malicious activities.

In the future, the security community should encourage collaboration and data sharing among researchers, network administrators, and security experts, which can facilitate the collection of diverse and representative traffic data. Besides, developing techniques for generating realistic synthetic data is also helpful. It can augment existing traffic datasets and provide additional insights into network dynamics and attack patterns.

It is particularly important to note that background or benign traffic data also plays a crucial role in the development and evaluation of FGTA-based detection methods. The presence of diverse and representative benign traffic data allows researchers and developers to assess the performance of their detection methods more accurately, taking into consideration the wide range of legitimate network activities. By incorporating realistic benign traffic patterns in the development and evaluation process, FGTA methods can be better tailored to minimize false positives and maximize detection accuracy in real-world scenarios.

7.5 Standardizing the Evaluation Pipeline

Another important future research direction is to establish a standardized evaluation pipeline, which is essential for assessing the performance, robustness, and generalizability of FGTA methods across different network environments and attack scenarios. A standardized evaluation pipeline also enables researchers and developers to compare various FGTA techniques objectively and identify the most effective approaches for specific use cases.

To accomplish this, researchers and developers can establish benchmarking frameworks and leaderboards for FGTA methods, promoting competition and innovation within the field. By offering a platform to compare the performance of various techniques, researchers can identify best practices and cutting-edge methods for detecting and mitigating malicious activities in network environments. Furthermore, akin to ImageNet in computer vision [126], creating and utilizing shared datasets for evaluating FGTA methods can enable fair and unbiased comparisons among different techniques. Adopting a consistent set of performance metrics and evaluation protocols can also contribute to the standardization of the evaluation process, facilitating the comparison of FGTA methods.

7.6 Integrating FGTA into Other Analytical Systems

While FGTA methods effectively detect various types of malicious activities, their capacity to deduce accurate application-layer information is constrained by the coarse-grained nature of the input traffic data. Addressing this challenge involves integrating FGTA methods with other analytical systems to enhance their performance and capabilities. Some potential systems for integration include:

1. **Endpoint-based monitoring systems:** As demonstrated in Chapter VI, integrating FGTA methods with endpoint-based monitoring systems can significantly improve the efficacy of L7 DDoS defense. By combining the strengths of both approaches, researchers can develop more comprehensive and accurate detection methods. In the future, researchers can also explore such integration to tackle other network security challenges, such as malware detection and intrusion detection.

2. **Threat intelligence systems [384]:** Incorporating threat intelligence data into FGTA methods can provide valuable context for identifying and understanding malicious activities. By leveraging information on known threats, attack patterns, and threat actors, FGTA methods can be enhanced to detect and mitigate a broader range of attacks.

3. **Security information and event management (SIEM) systems [67]:** SIEM systems collect, analyze, and correlate security event data from multiple sources to detect and respond to security incidents. Integrating FGTA methods with SIEM systems can provide a holistic view of network security and enable more effective detection and mitigation of malicious activities.

By exploring these integration opportunities, researchers can further expand the capabilities of FGTA-based methods and enhance their performance in detecting and mitigating malicious activities across various network environments.

CHAPTER VIII
CONCLUSION

TA is a commonly employed technique that captures and utilizes network features from the network or transport layers to infer information about ongoing communications. It is extensively utilized to detect a variety of malicious activities within the network. However, TA-based methods are encountering escalating challenges in detecting advanced malicious activities and providing fine-grained detection results that are necessary for further action. This highlights the need for more sophisticated and nuanced approaches to handle the evolving landscape of cybersecurity threats.

In this dissertation, our objective is to bridge this gap by introducing several innovative FGTA approaches designed for fine-grained detection of various malicious activities within the network. We concentrate on identifying hard-to-detect malicious activities, broadening the use cases of FGTA, enhancing the system usability, and merging network traffic data with endpoint information to enhance application-layer situational awareness. These contributions not only push the boundaries of the field but also tackle some of the urgent issues presently confronted by traditional TA methodologies.

Throughout this dissertation, we deliver key contributions pertaining to FGTA for the detection of malicious activities, showcased through the following five projects:

– We conduct a comprehensive survey of existing FGTA approaches, identifying the gaps and opportunities in the current state of the art. This survey not only provides a systematic understanding of the field but also serves as a roadmap for future research directions.

- To broaden the use cases of FGTA, we propose CJ-Sniffer, a privacy-aware cryptojacking detection approach that efficiently detects cryptojacking traffic by accessing anonymized, content-agnostic metadata of network traffic from the gateway of the network. To our best knowledge, CJ-Sniffer is the first approach to distinguish cryptojacking traffic from user-initiated cryptocurrency mining traffic, allowing for fine-grained traffic discrimination and improved protection against this growing threat.

- We further expand the application range of FGTA by introducing BotFlowMon, a learning-based, content-agnostic approach for detecting OSN bot traffic, relying only on flow-level traffic data as input and utilizing novel algorithms and techniques to classify OSN bot traffic from real OSN user traffic. This innovative approach overcomes the limitations of traditional social bot detection methods, which typically require private payload information, social relationships, or activity histories.

- We showcase that FGTA can be made more usable, as illustrated by our novel learning-based DDoS detection approach that underscores both explainability and adaptability. This method utilizes an enhanced KNN algorithm and classifies DDoS sources with fine granularity. It not only exhibits high accuracy and efficiency in detecting DDoS attacks but also provides explanatory information. This feature enables network administrators to easily scrutinize detection results and make necessary interventions. Furthermore, it can adapt to new network environments without the need for model retraining.

– In the end, we present a reinforcement learning-based defense approach against L7 DDoS attacks, which combines FGTA and endpoint-based defense approaches for better application-layer situational awareness. It actively monitors and analyzes the victim server and applies different strategies under different conditions to protect the server while minimizing collateral damage to legitimate requests. This approach addresses the challenge of detecting and defending against L7 DDoS attacks, which are difficult for traditional TA-based methods due to their limited visibility of transport and network layers.

Our evaluation results demonstrate that the proposed FGTA approaches achieve high accuracy and efficiency in detecting and mitigating various malicious activities, while maintaining privacy-preserving features, providing explainable and adaptable results, or providing comprehensive application-layer situation awareness. These contributions significantly advance both the fields of FGTA and malicious activity detection. This dissertation can serve as a foundation for future research and development in these critical areas.

Although our work has addressed several pressing challenges, there remain areas for further investigation and improvement. Future research could explore the development of even more advanced FGTA techniques, the application of these techniques to other malicious activities beyond those covered in this dissertation, and the enhancement of usabilities in FGTA-based solutions. Additionally, further exploration of the integration of network traffic data with endpoint information to provide more comprehensive protection and situational awareness is warranted. By building on the foundation laid in this dissertation, we believe that the field of malicious activity detection will continue to evolve and adapt to the ever-changing landscape of a variety of threats.

APPENDIX A

EXTENSION FOR THE DENSITY-VALLEY-BASED CLUSTERING

Algorithm 2 only applies to one-dimension datasets since the valley point competition mechanism only handles circumstances where two clusters share a valley point. This design is sufficient for processing transaction fingerprint data, but not able to deal with multi-dimension datasets where there can be more than two clusters sharing a valley point. We thus designed a more universal valley point competition mechanism in Algorithm 5 to solve this problem. In particular, if data point e in Algorithm 2 is a valley point of *multiple* clusters $\{c_1, c_2, ..., c_n\}$, it can invoke Algorithm 5 instead. It first divides competing clusters into a low-density group and a high-density group. It then merges all the clusters in the low-density group, as well as the cluster with the lowest density in the high-density group, into one cluster, and assigns valley point e to the cluster with the fewest data points among all clusters. With the help of Algorithm 5, Algorithm 2 can handle more complicated transaction fingerprint configurations, or even serve as a general clustering algorithm to process other types of high-dimension and low-dimension datasets.

Algorithm 5 Valley point competition for $n \geq 2$ clusters.

```
1: Input: Valley point $e$, set of competing clusters $C_n = \{c_1, c_2, ..., c_n\}$, set $C$ from Algorithm 2 that stores current clusters, valley point index $\rho$
2: $C_{low} = \phi$; $C_{high} = \phi$    ▷ two sets to store the low density and high density clusters from $C_n$, respectively
3: $C = C$ - $C_n$    ▷ remove $C_n$ from $C$
4: for cluster $c$ in $C_n$ do
5:     if $c.density \cdot \rho \leq e.density$ then
6:         Add $c$ to $C_{low}$
7:     else
8:         Add $c$ to $C_{high}$
9:     end if
10: end for
11: if $C_{low} \neq \phi$ then
12:     if $|C_{high}| >= 2$ then
13:         $c_{border}$ = the cluster with the lowest density in $C_{high}$
14:         remove $c_{border}$ from $C_{high}$
15:         $c_{new} = merge(C_{low}, \{c_{border}\})$
16:         $C_{all} = C_{high} \cup \{c_{new}\}$
17:         Add $e$ to the cluster with the fewest data points in $C_{all}$
18:         Add $C_{all}$ to $C$
19:     else
20:         $c_{new} = merge(C_{low}, C_{high})$
21:         Add $e$ to cluster $c_{new}$
22:         Add $c_{new}$ to $C$
23:     end if
24: else
25:     Add $e$ to the cluster with the fewest data points in $C_{high}$
26:     Add $C_{high}$ to $C$
27: end if
28: return $C$
```

APPENDIX B

SOCIAL BOTS SIMULATION

There are various types of social bots existing in the world, also, they can create different traffic flows during operations. BotFlowMon's design requires inputting a significant amount of labeled traffic data as ground truth. To derive the labeled dataset, we categorized the social bots into the following five types and comprehensively simulated them.

B.1 Chatbot

Chatbots are active in messaging applications such as Twitter Direct Messages, Facebook Messenger, or WeChat. Either artificial-intelligence-powered or merely logic-based, they can automatically perform conversations with regular users.

The simulation of chatbots relies on existing widely used chatbot frameworks, APIs, and open-source programs such as botmaster [386], Ontbot [43], BotLibre [30], and python-twitter API. We created many Twitter and Facebook bot accounts only for research purposes under the OSNs' terms and conditions of service. The chatbots we created only talked to the recruited participants during the data generation process. We collected multitudinous traffic flows of the conversations between real users and the chatbots, with different frequencies of interaction, response time and transmission content (image, audio, text, or hyperlink).

B.2 Post Bot

In part due to the easily usable official and third-party APIs, poster bots have become the most common social bots in OSN. They distribute spam tweets and Facebook posts with malicious URLs in most cases or malicious texts occasionally [440] [182].

In order to simulate post bots, we downloaded several popular open-source poster bot software from GitHub [22] and wrote some poster bot programs based on APIs such as Tweepy [326] and Facebook API [181]. The open-source programs we used included botmaster [386], BotLibre [30], and TwitterBot [31]. We first set up OSN accounts on Twitter and Facebook platforms. Then, we ran these bots programs with different frequencies, including random frequencies, during different time periods to post some harmless messages that contained texts, videos, images and external URLs on Twitter and Facebook.

B.3 Amplification Bot

Amplification bot, due to the large volume of messages it can generate, is often used to create hot topics for commercial promotion, consensus manipulation, and spam distribution. Without creating new content, amplification bots often work as fake followers, such as those Twitter or Facebook accounts specifically created to inflate the number of followers of a target account. Amplification bots also can serve as forwarding and liking robots, popularize junk information, and help commercial promotion.

Since the social relationship is unknown in NetFlow data, we only need to simulate every individual amplification bot's interactions with OSNs. The simulation of amplification bots is similar to the post bot simulation. We used open-sourced programs [30, 31, 386] and implemented API-based bot scripts to simulate amplification bots. Besides, we used OAuth [188] software for token management and switching accounts.

B.4 OSN Crawler

OSN crawlers can exploit OSNs to aggregate data of a large number of OSN users for re-publication or other nefarious purposes that violate users' privacy and security. There are two types of OSN crawlers and we simulated both of them. One is API-based, which relies on a large botnet to extract users' private, sensitive data. Twitter and Facebook used to have powerful APIs such that even a small number of accounts can fetch a large amount of information from their sites. Recently OSN operators have taken plenty of measures to limit the APIs' capacity in crawling user data, especially after the Facebook-Cambridge Analytica data scandal [24]. Moreover, in OSNs it is common that a user's information can only be accessed by their friends or followers. Therefore, a significant amount of social bots are necessary to overcome the API limitations and retrieve users' private information. The following code exemplifies a crawling process.

```
import twitter
api = twitter.Api()
api.GetUser(user)
api.GetFollowers()
api.GetStatus(status_id)
api.GetFriends(user)
api.LookupFriendship(user)
```

Another type of OSN crawler is page crawler. Instead of using API, it reads the HTML files of OSNs and uses regular expressions to extract target information. The NetFlow traffic of this bot has a strong resemblance to a regular user's traffic but still differs in key aspects, such as flow density and operation frequency.

B.5 Hybrid Bot

The hybrid bot is not a specific type of social bot. Instead, it is an arbitrary combination of different types of social bots. This characteristic makes the activities of a hybrid bot hard to detect. However, with BotFlowMon's subdivision module, we can subdivide its transactions into actions, thus still able to detect it.

We implemented a hybrid bot that consists of a chatbot, a post bot, an amplification bot, and an OSN crawler on a single node. During the execution of the hybrid bot, its chatbot is always active, so it automatically replies whenever other accounts send it messages. Meanwhile, it randomly conducts actions provided by other underlying bots. We simulated its activities with both periodic and random frequencies.

APPENDIX C

ETHICAL AND HUMAN SUBJECT ISSUES OF BOTFLOWMON

We carefully addressed the ethical and human subject issues involved in the research of OSN bot traffic detection. As the development and evaluation of BotFlowMon involved human subjects, we obtained the Institutional Review Board (IRB) approval to ensure that our procedures are ethical and appropriate in order to safeguard the welfare of the campus network and the privacy of research participants. We introduce the main measures we adopted to address the ethical considerations below.

C.1 Data Generation

In recruiting participants, we adopted an informed consent procedure, which provided the potential participants with a clear description about the project before they agree. After the data generation, all their personal data were deleted immediately. In simulating social bots, we ensured they followed the terms of service of OSN providers, were harmless to other users and the OSN environments, and left no traces after each experiment. For example, we used weather reports, university announcements, and encyclopedic knowledge to simulate fraud and spam posts for post and amplification bots. The external URLs contained in the bot messages were well-known innocent URLs, such as google.com and facebook.com. The chatbots and OSN crawlers only interacted with the informed participants, and the personal data gathered by them were not stored in any form.

C.2 Data Collection

All the data we collected were content-agnostic NetFlow records, along with the labels, time stamps, and short descriptions of real user and social bot activities. No payload or content data were downloaded and stored during the process. For the NetFlow records from the campus network, every internal IP address was anonymized, so it cannot be mapped back to any individual. Using anonymous IP addresses, however, makes it hard to identify the NetFlow records created by ourselves. To address this issue, every time we generated legitimate or illegitimate traffic, we pinged several external IP addresses and recorded their time stamps to create distinctive features of these traffic, making it easy to identify their NetFlow records and discard all irrelevant ones. In addition, all the data we collected were stored in an encrypted form, and any access to the data must have the approval of the principal investigator of this project.

APPENDIX D
TIME COMPLEXITY ANALYSIS FOR THE PROPOSED DENSITY-VALLEY-BASED CLUSTERING ALGORITHM

D.1 Time Complexities for Different Modules

Preprocessing module: This module's input is all the NetFlow records from a monitored network. Assume their number is n. This module applies the longest prefix match to output OSN-related NetFlow records, which takes a linear time. Its time complexity is therefore $O(n)$.

Flow aggregation module: This module's input is OSN-related NetFlow records per OSN account (which can be either a real user or a social bot). Assume the number of time bins of these records is m. As this module uses DBSCAN to aggregate related NetFlow records into transactions, it takes $O(mlogm)$ time.

Transaction fingerprint generation module: This module's input is a set of NetFlow records that form a single transaction. Its purpose is to produce an $f \times N$ matrix as the fingerprint of the transaction. This module's time complexity is thus $O(fN)$.

Transaction subdivision module: This module's input is the $f \times N$ fingerprint matrix of a single transaction. It applies its density-valley-based clustering algorithm to subdivide the transaction and output a set of actions in the form of action fingerprint matrixes. As the time complexity of the clustering algorithm is similar to DBSCAN, this module's time complexity is $O(NlogN)$.

Machine learning & classification module: This module's input is j actions that a transaction is subdivided into. Assuming the number of layers in CNN or MLP classification model is l, this module is to classify each action with a time complexity $O(l)$. This module's time complexity is thus $O(lj)$.

D.2 Overall Time Complexity

As BotFlowMon processes incoming NetFlow records online, its time complexity is the sum of the time complexity of every module. Assume after the preprocessing module the selected NetFlow records belong to a OSN accounts. Also assume the flow aggregation module outputs b transactions per OSN account. BotFlowMon's time complexity is thus $O(n)$ + $a{\cdot}O(mlogm)$ + $a{\cdot}b{\cdot}O(fN)$ + $a{\cdot}b{\cdot}O(NlogN)$ + $a{\cdot}b{\cdot}O(lj)$. Here, m is a constant as BotFlowMon receives NetFlow records periodically, f and N are constants as f is 4 or 6 and N is 200 in our setup, and l is a constant with a default value of 9 for CNN and 7 for MLP. So BotFlowMon's time complexity can be simplified as $O(n)$+$O(a)$+$O(ab)$+$O(ab)$+$O(abj)$, which is equivalent to $O(n)$+$O(abj)$. Note abj represents all the actions by all OSN accounts from these n NetFlow records, which is approximately proportional to n. Therefore, BotFlowMon's time complexity is $O(n)$, where n is the total number of input NetFlow records.

REFERENCES CITED

[1] Arbor threat mitigation system. `https://www.netscout.com/product/arbor-threat-mitigation-system`. Accessed: 2021-1-11.

[2] Argus project. `https://openargus.org/`. Date of visit: 2021-11-25.

[3] Argus: the first network flow system. `https://openargus.org/`.

[4] Data sheet of sniffer infinistream. `http://mavin.com/pictures/InfiniStream.pdf`. Accessed: 2022-02-10.

[5] The Front Range GigaPoP: Connecting colorado and wyoming with hundreds of gigabits. `https://frgp.net/frgp/index.shtml`.

[6] Kentik is network observability. `https://www.kentik.com/`. Accessed: 2021-1-11.

[7] Mininet: An instant virtual network on your laptop (or other pc). `http://mininet.org/`. Accessed: 2020-12-11.

[8] Netflow v5 formats. `https://www.ibm.com/docs/en/npi/1.3.0?topic=versions-netflow-v5-formats`. Accessed: 2022-03-14.

[9] Shadowsocks - a fast tunnel proxy that helps you bypass firewalls. `https://shadowsocks.org/`. Accessed: 2022-02-11.

[10] Solaris snoop packet sniffer. `http://www.softpanorama.org/Net/Sniffers/snoop.shtml`. Accessed: 2022-02-10.

[11] Tcpdump & libpcap. `https://www.tcpdump.org/`. Accessed: 2022-02-10.

[12] The CAIDA UCSD "DDoS attack 2007" dataset. `https://www.caida.org/data/passive/ddos-20070804_dataset.xml`, 2007. CAIDA.

[13] DARPA 2009 intrusion detection dataset (colorado state university). `http://www.darpa2009.netsec.colostate.edu/`, 2009.

[14] National cyber awareness system: Security tip - understanding Denial-of-Service Attacks. `https://www.us-cert.gov/ncas/tips/ST04-015`, 2009. US-Cert.

[15] FRGP ntp flow data - ntp reflection attack.
https://www.impactcybertrust.org/dataset_view?idDataset=776, 2014. University of Southern California-Information Sciences Institute.

[16] Keras. https://keras.io, 2015.

[17] TensorFlow: Large-scale machine learning on heterogeneous systems.
https://www.tensorflow.org/, 2015.

[18] DDoS Chargen 2016 dataset - Internet traffic data containing a DDoS attack based on UDP Chargen protocol.
https://www.impactcybertrust.org/dataset_view?idDataset=693, 2016. Merit Network, Inc.

[19] Coinimp: Free javascript mining - browser mining.
https://www.coinimp.com/, 2017. Accessed: 2021-04-15.

[20] XMRig: high performance, open source, cross platform xmr miner.
https://github.com/xmrig/xmrig, 2017. Accessed: 2021-04-15.

[21] Easy pool miner. https://jefreesujit.github.io/easyminer/, 2018. Accessed: 2021-04-15.

[22] Poster bots from GitHub. https://github.com/search?q=poster+bot, 2018.

[23] Attackers use large-scale bots to launch attacks on social media platforms.
https://www.helpnetsecurity.com/2019/08/27/attacks-on-social-media-platforms/, 2019. Help Net Security.

[24] Facebook-Cambridge Analytica data scandal. https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal, 2019.

[25] Minero: Monero miner for web browsers. https://minero.cc/, 2019. Accessed: 2021-04-15.

[26] New tor release: Tor 0.4.0.5.
https://blog.torproject.org/new-release-tor-0405/, 2019. Accessed: 2022-02-15.

[27] ProfiShark Network TAPs.
https://www.profitap.com/profishark-network-taps/, 2019. Accessed: 2022-02-11.

[28] Webminepool: Multifunctional mining service for site owners and individuals.
https://www.webminepool.com/, 2019. Accessed: 2021-04-15.

[29] Cisco annual internet report (2018–2023) white paper. `https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report`, 2020.

[30] The open source chatbot and artificial intelligence platform, 2020.

[31] Twitter bot made with Golang. `https://github.com/benoitletondor/TwitterBot`, 2020.

[32] The history of ethereum. `https://ethereum.org/en/history/`, 12 2021. Accessed: 2022-4-6.

[33] PF-RING: High-speed packet capture, filtering and analysis. `https://www.ntop.org/products/packet-capture/pf_ring/`, 2021. Accessed: 2021-05.

[34] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *Osdi* (2016), vol. 16, Savannah, GA, USA, pp. 265–283.

[35] Abe, K., and Goto, S. Fingerprinting attack on tor anonymity using deep learning. *Proceedings of the Asia-Pacific Advanced Network 42* (2016), 15–20.

[36] Aceto, G., Ciuonzo, D., Montieri, A., and Pescapé, A. Multi-classification approaches for classifying mobile app traffic. *Journal of Network and Computer Applications 103* (2018), 131–145.

[37] Aceto, G., Ciuonzo, D., Montieri, A., and Pescapè, A. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer networks 165* (2019), 106944.

[38] Aceto, G., Ciuonzo, D., Montieri, A., and Pescapé, A. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management 16*, 2 (2019), 445–458.

[39] Agboma, F., Smy, M., and Liotta, A. Qoe analysis of a peer-to-peer television system. In *Proceedings of IADISInt. Conf. on Telecommunications, Networks and Systems* (2008), pp. 365–382.

[40] Aiolli, F., Conti, M., Gangwal, A., and Polato, M. Mind your wallet's privacy: identifying bitcoin wallet apps and user's actions through network traffic analysis. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (2019), pp. 1484–1491.

[41] Ajao, O., Hong, J., and Liu, W. A survey of location inference techniques on twitter. *Journal of Information Science 41*, 6 (2015), 855–864.

[42] Akyildiz, I. F., Lee, A., Wang, P., Luo, M., and Chou, W. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks 71* (2014), 1–30.

[43] Al-Zubaide, H., and Issa, A. A. Ontbot: Ontology based chatbot. In *Fourth international symposium on innovation in information & communication technology* (2011), pp. 7–12.

[44] Alan, H. F., and Kaur, J. Can android applications be identified using only tcp/ip headers of their launch time traffic? In *Proceedings of the 9th ACM conference on security & privacy in wireless and mobile networks* (2016), pp. 61–66.

[45] Alex Mcdonnell, Nichols Mavis, S. R. L. G. M. J. R. A. S. Blocking cryptocurrency mining using cisco security products, 2019.

[46] Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., and Aljaaf, A. J. A systematic review on supervised and unsupervised machine learning algorithms for data science. *Supervised and unsupervised learning for data science* (2020), 3–21.

[47] Alon, N., and Spencer, J. H. *The probabilistic method.* John Wiley & Sons, 2016.

[48] Amoli, P. V., Hamalainen, T., David, G., Zolotukhin, M., and Mirzamohammad, M. Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets. *JDCTA (International Journal of Digital Content Technology and its Applications 10*, 2 (2016), 1–13.

[49] Anderson, B., and McGrew, D. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on knowledge discovery and data mining* (2017), pp. 1723–1732.

[50] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. OPTICS: ordering points to identify the clustering structure. *ACM Sigmod record 28*, 2 (1999), 49–60.

[51] Anthony, S. GitHub battles “largest DDoS” in site’s history, targeted at anti-censorship tools. `https://arstechnica.com/`, March 30, 2015.

[52] Ateniese, G., Hitaj, B., Mancini, L. V., Verde, N. V., and Villani, A. No place to hide that bytes won't reveal: Sniffing location-based encrypted traffic to track a user's position. In *International Conference on Network and System Security* (2015), Springer, pp. 46–59.

[53] Auld, T., Moore, A. W., and Gull, S. F. Bayesian neural networks for Internet traffic classification. *IEEE Transactions on Neural Networks 18*, 1 (2007), 223–239.

[54] Bagnulo, M., Matthews, P., and van Beijnum, I. Stateful nat64: Network address and protocol translation from ipv6 clients to ipv4 servers. Tech. rep., 2011.

[55] Barati, M., Abdullah, A., Udzir, N. I., Mahmod, R., and Mustapha, N. Distributed denial of service detection using hybrid machine learning technique. In *International Symposium on Biometrics and Security Technologies* (2014), IEEE, pp. 268–273.

[56] Barford, P., Kline, J., Plonka, D., and Ron, A. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), pp. 71–82.

[57] Barnum, S. Standardizing cyber threat intelligence information with the structured threat information expression (stix). *Mitre Corporation 11* (2012), 1–22.

[58] Barradas, D., Santos, N., Rodrigues, L., Signorello, S., Ramos, F. M., and Madeira, A. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA* (2021).

[59] Bashir, A., Huang, C., Nandy, B., and Seddigh, N. Classifying P2P activity in NetFlow records: A case study on BitTorrent. In *IEEE international conference on communications* (2013), pp. 3018–3023.

[60] Bawany, N. Z., Shamsi, J. A., and Salah, K. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering 42*, 2 (2017), 425–441.

[61] Benevenuto, F., Magno, G., Rodrigues, T., and Almeida, V. Detecting spammers on Twitter. In *Collaboration, electronic messaging, anti-abuse and spam conference* (2010), p. 12.

[62] Bernaille, L., and Teixeira, R. Early recognition of encrypted applications. In *International Conference on Passive and Active Network Measurement* (2007), Springer, pp. 165–175.

[63] Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., and Salamatian, K. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review 36*, 2 (2006), 23–26.

[64] Bernaille, L., Teixeira, R., and Salamatian, K. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference* (2006), pp. 1–12.

[65] Beznazwy, J., and Houmansadr, A. How china detects and blocks shadowsocks. In *Proceedings of the ACM Internet Measurement Conference* (2020), pp. 111–124.

[66] Bhat, S., Lu, D., Kwon, A., and Devadas, S. Var-cnn: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies 1*, 19.

[67] Bhatt, S., Manadhata, P. K., and Zomlot, L. The operational role of security information and event management systems. *IEEE security & Privacy 12*, 5 (2014), 35–41.

[68] Bhaya, W., and EbadyManaa, M. Ddos attack detection approach using an efficient cluster analysis in large data scale. In *Annual Conference on New Trends in Information & Communications Technology Applications* (2017), IEEE, pp. 168–173.

[69] Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials 16*, 1 (2013), 303–336.

[70] Bhuyan, M. H., Kashyap, H. J., Bhattacharyya, D. K., and Kalita, J. K. Detecting distributed denial of service attacks: methods, tools and future directions. *The Computer Journal 57*, 4 (2014), 537–556.

[71] Bijmans, H. L., Booij, T. M., and Doerr, C. Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale. In *28th {USENIX} Security Symposium ({USENIX} Security 19)* (2019), pp. 1627–1644.

[72] Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., and Kruegel, C. Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis. In *28th annual computer security applications conference* (2012), pp. 129–138.

[73] Bilge, L., Strufe, T., Balzarotti, D., and Kirda, E. All your contacts are belong to us: Automated identity theft attacks on social networks. In *18th international conference on world wide web* (2009).

[74] Bissias, G. D., Liberatore, M., Jensen, D., and Levine, B. N. Privacy vulnerabilities in encrypted http streams. In *International Workshop on Privacy Enhancing Technologies* (2005), Springer, pp. 1–11.

[75] Bowen, D., and Ungar, L. Generalized shap: Generating multiple types of explanations in machine learning. *arXiv preprint arXiv:2006.07155* (2020).

[76] Brandt, A. Compromised exchange server hosting cryptojacker targeting other exchange servers. https://news.sophos.com/en-us/2021/04/13/compromised-exchange-server-hosting-cryptojacker-targeting-other-exchange-servers/, 2021.

[77] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[78] Brownlee, N., Mills, C., and Ruth, G. Traffic flow measurement: Architecture. Tech. rep., RFC 2722, 1999.

[79] Brunella, M. S., Belocchi, G., Bonola, M., Pontarelli, S., Siracusano, G., Bianchi, G., Cammarano, A., Palumbo, A., Petrucci, L., and Bifulco, R. {hXDP}: Efficient software packet processing on {FPGA}{NICs}. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (2020), pp. 973–990.

[80] Cai, X., Nithyanand, R., and Johnson, R. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), pp. 121–130.

[81] Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 227–238.

[82] Cai, X., Zhang, X. C., Joshi, B., and Johnson, R. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 605–616.

[83] Callado, A., Kamienski, C., Szabo, G., Gero, B. P., Kelner, J., Fernandes, S., and Sadok, D. A survey on internet traffic identification. *IEEE Communications Surveys & Tutorials 11*, 3 (2009), 37–52.

[84] Cao, Q., Sirivianos, M., Yang, X., and Pregueiro, T. Aiding the detection of fake accounts in large scale social online services. In *9th USENIX conference on networked systems design and implementation* (2012), pp. 197–210.

[85] Cao, Q., and Yang, X. SybilFence: Improving social-graph-based Sybil defenses with user negative feedback. *arXiv preprint arXiv:1304.3819* (2013).

[86] Cao, Y., Jiang, H., Deng, Y., Wu, J., Zhou, P., and Luo, W. Detecting and mitigating ddos attacks in sdn using spatial-temporal graph convolutional network. *IEEE Transactions on Dependable and Secure Computing 19*, 6 (2021), 3855–3872.

[87] Caprolu, M., Raponi, S., Oligeri, G., and Di Pietro, R. Cryptomining makes noise: a machine learning approach for cryptojacking detection. *arXiv preprint arXiv:1910.09272* (2019).

[88] Caprolu, M., Raponi, S., Oligeri, G., and Di Pietro, R. Cryptomining makes noise: Detecting cryptojacking via machine learning. *Computer Communications 171* (2021), 126–139.

[89] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 1–58.

[90] Chandrasekaran, B. Survey of network traffic models. *Waschington University in St. Louis CSE 567* (2009).

[91] Chen, C.-L. A new detection method for distributed denial-of-service attack traffic based on statistical test. *Journal of Universal Computer Science 15*, 2 (2009), 488–504.

[92] Chen, Y., Hwang, K., and Ku, W.-S. Collaborative detection of ddos attacks over multiple network domains. *IEEE Transactions on Parallel and Distributed Systems 18*, 12 (2007), 1649–1662.

[93] Chen, Y.-C., Liao, Y., Baldi, M., Lee, S.-J., and Qiu, L. Os fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), pp. 173–180.

[94] Chen, Z., He, K., Li, J., and Geng, Y. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International conference on big data (big data)* (2017), IEEE, pp. 1271–1276.

[95] Cherubin, G., Hayes, J., and Juárez, M. Website fingerprinting defenses at the application layer. *Proc. Priv. Enhancing Technol. 2017*, 2 (2017), 186–203.

[96] Choi, B.-Y., and Bhattacharyya, S. On the accuracy and overhead of cisco sampled netflow. In *Proceedings of ACM SIGMETRICS Workshop on Large Scale Network Inference (LSNI)* (2005), pp. 1–6.

[97] Choi, H., Lee, H., Lee, H., and Kim, H. Botnet detection by monitoring group activities in DNS traffic. In *7th IEEE international conference on computer and information technology* (2007), pp. 715–720.

[98] Chollet, F., et al. Keras, 2015.

[99] Choorod, P., and Weir, G. Tor traffic classification based on encrypted payload characteristics. In *2021 National Computing Colleges Conference (NCCC)* (2021), IEEE, pp. 1–6.

[100] Choraś, M., Pawlicki, M., Puchalski, D., and Kozik, R. Machine learning–the results are not the only thing that matters! what about security, explainability and fairness? In *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV 20* (2020), Springer, pp. 615–628.

[101] Chu, Z., Gianvecchio, S., Wang, H., and Jajodia, S. Detecting automation of Twitter accounts: Are you a human, bot, or cyborg? *IEEE Transactions on Dependable and Secure Computing 9*, 6 (2012), 811–824.

[102] Claise, B. Cisco Systems NetFlow services export version 9. RFC 3954, 2004.

[103] Claise, B., Trammell, B., and Aitken, P. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information, 2013.

[104] Cloudflare. Application layer DDoS attack. `https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/`.

[105] Comer, D. Ubiquitous b-tree. *ACM Computing Surveys (CSUR) 11*, 2 (1979), 121–137.

[106] Cong, L. W., He, Z., and Li, J. Decentralized mining in centralized pools. *The Review of Financial Studies 34*, 3 (2021), 1191–1235.

[107] Conti, M., Li, Q. Q., Maragno, A., and Spolaor, R. The dark side(-channel) of mobile devices: A survey on network traffic analysis. *IEEE Communications Surveys & Tutorials 20*, 4 (2018), 2658–2713.

[108] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security 11*, 1 (2015), 114–125.

[109] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (2015), pp. 297–304.

[110] Continella, A., Fratantonio, Y., Lindorfer, M., Puccetti, A., Zand, A., Kruegel, C., and Vigna, G. Obfuscation-resilient privacy leak detection for mobile apps through differential analysis. In *NDSS* (2017).

[111] Coull, S. E., and Dyer, K. P. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review 44*, 5 (2014), 5–11.

[112] Coulter, R., Han, Q.-L., Pan, L., Zhang, J., and Xiang, Y. Data-driven cyber security in perspective—intelligent traffic analysis. *IEEE transactions on cybernetics 50*, 7 (2019), 3081–3093.

[113] Crotti, M., Dusi, M., Gringoli, F., and Salgarelli, L. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review 37*, 1 (2007), 5–16.

[114] Cunha, Í., Silveira, F., Oliveira, R., Teixeira, R., and Diot, C. Uncovering artifacts of flow measurement tools. In *International Conference on Passive and Active Network Measurement* (2009), Springer, pp. 187–196.

[115] Danezis, G., and Mittal, P. SybilInfer: Detecting Sybil nodes using social networks. In *The network and distributed system security symposium* (2009), pp. 1–15.

[116] Darabian, H., Homayounoot, S., Dehghantanha, A., Hashemi, S., Karimipour, H., Parizi, R. M., and Choo, K.-K. R. Detecting cryptomining malware: a deep learning approach for static and dynamic analysis. *Journal of Grid Computing* (2020), 1–11.

[117] Das, A. K., Pathak, P. H., Chuah, C.-N., and Mohapatra, P. Contextual localization through network traffic analysis. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications* (2014), IEEE, pp. 925–933.

[118] Das, A. K., Pathak, P. H., Chuah, C.-N., and Mohapatra, P. Privacy-aware contextual localization using network traffic analysis. *Computer Networks 118* (2017), 24–36.

[119] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry* (2004), ACM, pp. 253–262.

[120] Davis, C. A., Varol, O., Ferrara, E., Flammini, A., and Menczer, F. Botornot: A system to evaluate social bots. In *25th international conference companion on world wide web* (2016), pp. 273–274.

[121] Davis, M. H. *Markov models & optimization*, vol. 49. CRC Press, 1993.

[122] De la Cadena, W., Mitseva, A., Hiller, J., Pennekamp, J., Reuter, S., Filter, J., Engel, T., Wehrle, K., and Panchenko, A. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020), pp. 1971–1985.

[123] DEBRYUNE, Dnaleor, and project, M. PoW change and key reuse. `https://www.getmonero.org/2018/02/11/PoW-change-and-key-reuse.html`, 2 2018. Accessed: 2022-4-6.

[124] Delaney, D. How to detect cryptocurrency mining activity on your network. `https://www.netfort.com/blog/detect-cryptocurrency-mining-activity/`, 2018.

[125] Demuth, H. B., Beale, M. H., De Jess, O., and Hagan, M. T. *Neural network design.* Martin Hagan, 2014.

[126] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.

[127] Deng, Z., Liu, Z., Chen, Z., and Guo, Y. The random forest based detection of shadowsock's traffic. In *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)* (2017), vol. 2, IEEE, pp. 75–78.

[128] Difallah, D. E., Demartini, G., and Cudré-Mauroux, P. Mechanical cheat: Spamming schemes and adversarial techniques on crowdsourcing platforms. In *CrowdSearch* (2012), pp. 26–30.

[129] Dimopoulos, G., Leontiadis, I., Barlet-Ros, P., and Papagiannaki, K. Measuring video qoe from encrypted traffic. In *Proceedings of the 2016 Internet Measurement Conference* (2016), pp. 513–526.

[130] Dingledine, R., and Mathewson, N. Tor protocol specification. `https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt`. Date of visit: 2021-10-05.

[131] Doshi, R., Apthorpe, N., and Feamster, N. Machine learning ddos detection for consumer internet of things devices. In *IEEE Security and Privacy Workshops* (2018), pp. 29–35.

[132] Došilović, F. K., Brčić, M., and Hlupić, N. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)* (2018), IEEE, pp. 0210–0215.

[133] Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I., and Ghorbani, A. A. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)* (2016), sn, pp. 407–414.

[134] Duch, W., Setiono, R., and Zurada, J. M. Computational intelligence methods for rule-based data understanding. *Proceedings of the IEEE 92*, 5 (2004), 771–805.

[135] Dunteman, G. H. *Principal components analysis*. No. 69. Sage, 1989.

[136] Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy* (2012), IEEE, pp. 332–346.

[137] D'Alconzo, A., Drago, I., Morichetta, A., Mellia, M., and Casas, P. A survey on big data for network traffic monitoring and analysis. *IEEE Transactions on Network and Service Management 16*, 3 (2019), 800–813.

[138] Emmert-Streib, F., Yli-Harja, O., and Dehmer, M. Explainable artificial intelligence and machine learning: A reality rooted perspective. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10*, 6 (2020), e1368.

[139] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS) 32*, 2 (2014), 1–29.

[140] Erman, J., Arlitt, M., and Mahanti, A. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data* (2006), pp. 281–286.

[141] Erramilli, A., Roughan, M., Veitch, D., and Willinger, W. Self-similar traffic and network dynamics. *Proceedings of the IEEE 90*, 5 (2002), 800–819.

[142] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining* (1996), pp. 226–231.

[143] Fadlullah, Z. M., Taleb, T., Ansari, N., Hashimoto, K., Miyake, Y., Nemoto, Y., and Kato, N. Combating against attacks on encrypted protocols. In *2007 IEEE International Conference on Communications* (2007), IEEE, pp. 1211–1216.

[144] Feily, M., Shahrestani, A., and Ramadass, S. A survey of botnet and botnet detection. In *Third international conference on emerging security information, systems and technologies* (2009), pp. 268–273.

[145] Feng, Y. Botflowmon: Identify social bot traffic with netflow and machine learning.

[146] Feng, Y. Towards intelligent defense against application-layer ddos with reinforcement learning.

[147] Feng, Y. Packet-level cryptomining network traffic dataset. `https://github.com/yebof/CJ-Sniffer-Dataset`, 2022.

[148] Feng, Y. Toward finer granularity analysis of network traffic. Area exam, Mar. 2022.

[149] Feng, Y., and Li, J. Toward explainable and adaptable detection and classification of distributed denial-of-service attacks. In *2020 KDD MLHat: The First International Workshop on Deployable Machine Learning for Security Defense* (Cham, 2020), Springer International Publishing, pp. 105–121.

[150] Feng, Y., and Li, J. Towards explicable and adaptive ddos traffic classification. In *The 21st Passive and Active Measurement Conference-Poster* (2020).

[151] Feng, Y., Li, J., Jiao, L., and Wu, X. BotFlowMon: Learning-based, content-agnostic identification of social bot traffic flows. In *2019 IEEE Conference on Communications and Network Security (CNS)* (June 2019), pp. 169–177.

[152] Feng, Y., Li, J., Jiao, L., and Wu, X. Towards learning-based, content-agnostic detection of social bot traffic. *IEEE Transactions on Dependable and Secure Computing (TDSC) 18*, 5 (2021), 2149–2163.

[153] Feng, Y., Li, J., and Mirkovic, J. Unmasking the internet: A survey of fine-grained network traffic analysis.

[154] Feng, Y., Li, J., and Nguyen, T. Application-layer DDoS defense with reinforcement learning. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)* (June 2020), IEEE, pp. 1–10.

[155] Feng, Y., Li, J., and Sisodia, D. Cj-sniffer: Measurement and content-agnostic detection of cryptojacking traffic. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* (New York, NY, USA, Oct. 2022), RAID '22, Association for Computing Machinery, p. 482–494.

[156] Feng, Y., Luo, J., Ma, C., Li, T., and Hui, L. I can still observe you: Flow-level behavior fingerprinting for online social network. In *GLOBECOM 2022-2022 IEEE Global Communications Conference* (2022), IEEE, pp. 6427–6432.

[157] Feng, Y., Sisodia, D., and Li, J. Poster: Content-agnostic identification of cryptojacking in network traffic. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security* (2020), pp. 907–909.

[158] Feng, Y., Xu, J., and Weymouth, L. University blockchain research initiative (ubri): Boosting blockchain education and research. *IEEE Potentials 41*, 6 (2022), 19–25.

[159] Ferrara, E. Manipulation and abuse on social media. *ACM SIGWEB Newsletter*, Spring (2015), 1–9.

[160] Ferrara, E., Varol, O., Davis, C., Menczer, F., and Flammini, A. The rise of social bots. *Communications of the ACM 59*, 7 (2016), 96–104.

[161] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Rfc2616: Hypertext transfer protocol–http/1.1, 1999.

[162] Finsterbusch, M., Richter, C., Rocha, E., Muller, J.-A., and Hanssgen, K. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials 16*, 2 (2014), 1135–1156.

[163] Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B., and Peteiro-Barral, D. Online machine learning. In *Efficiency and Scalability Methods for Computational Intellect*. IGI global, 2013, pp. 27–54.

[164] Fotouhi, A., Qiang, H., Ding, M., Hassan, M., Giordano, L. G., Garcia-Rodriguez, A., and Yuan, J. Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. *IEEE Communications Surveys & Tutorials 21*, 4 (2019), 3417–3442.

[165] Friedman, J. H. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery 1*, 1 (1997), 55–77.

[166] Fu, Y., Xiong, H., Lu, X., Yang, J., and Chen, C. Service usage classification with encrypted internet traffic in mobile messaging apps. *IEEE Transactions on Mobile Computing 15*, 11 (2016), 2851–2864.

[167] Gagniuc, P. A. *Markov chains: from theory to implementation and experimentation.* John Wiley & Sons, 2017.

[168] Gallagher, A., Joshi, D., Yu, J., and Luo, J. Geo-location inference from image content and user tags. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (2009), IEEE, pp. 55–62.

[169] Gao, H., Chen, Y., Lee, K., Palsetia, D., and Choudhary, A. N. Towards online spam filtering in social networks. In *The network and distributed system security symposium* (2012), pp. 1–16.

[170] Gao, H., Hu, J., Wilson, C., Li, Z., Chen, Y., and Zhao, B. Y. Detecting and characterizing social spam campaigns. In *10th ACM Internet measurement conference* (2010), pp. 35–47.

[171] Gao, P., Wang, B., Gong, N. Z., Kulkarni, S. R., Thomas, K., and Mittal, P. SybilFuse: Combining local attributes with global structure to perform robust Sybil detection. In *IEEE conference on communications and network security* (2018), pp. 1–9.

[172] Gardner, M. W., and Dorling, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment 32*, 14-15 (1998), 2627–2636.

[173] Garreau, D., and Luxburg, U. Explaining the explainer: A first theoretical analysis of lime. In *International Conference on Artificial Intelligence and Statistics* (2020), PMLR, pp. 1287–1296.

[174] Goldberg, I., and Wood, C. A. Network-based website fingerprinting. `https://datatracker.ietf.org/doc/html/draft-wood-privsec-wfattacks-00`, 2019.

[175] Gomes, F., and Correia, M. Cryptojacking detection with cpu usage metrics. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)* (2020), IEEE, pp. 1–10.

[176] Gomes, G., Dias, L., and Correia, M. Cryingjackpot: Network flows and performance counters against cryptojacking. In *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)* (2020), IEEE, pp. 1–10.

[177] Gomes, J. a. V., Inácio, P. R. M., Pereira, M., Freire, M. M., and Monteiro, P. P. Detection and classification of peer-to-peer traffic: A survey. *ACM Comput. Surv. 45*, 3 (jul 2013).

[178] Gong, J., and Wang, T. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)* (2020), pp. 717–734.

[179] Gong, N. Z., Frank, M., and Mittal, P. SybilBelief: A semi-supervised learning approach for structure-based Sybil detection. *IEEE Transactions on Information Forensics and Security 9*, 6 (2014), 976–987.

[180] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

[181] Graham, W. *Facebook API developers guide*. Infobase Publishing, 2008.

[182] Grier, C., Thomas, K., Paxson, V., and Zhang, M. @spam: The underground on 140 characters or less. In *17th ACM conference on computer and communications security* (2010), pp. 27–37.

[183] Gringoli, F., Salgarelli, L., Dusi, M., Cascarano, N., Risso, F., and Claffy, K. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review 39*, 5 (2009), 12–18.

[184] Gu, G., Perdisci, R., Zhang, J., Lee, W., et al. BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX security symposium* (2008), pp. 139–154.

[185] Guyon, I., Gunn, S., Nikravesh, M., and Zadeh, L. A. *Feature extraction: foundations and applications*, vol. 207. Springer, 2008.

[186] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*, 1 (2009), 10–18.

[187] Han, J., Owusu, E., Nguyen, L. T., Perrig, A., and Zhang, J. Accomplice: Location inference using accelerometers on smartphones. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)* (2012), IEEE, pp. 1–9.

[188] Hardt, D. The OAuth 2.0 authorization framework.

[189] Hare, C. Simple network management protocol (snmp)., 2011.

[190] Hartigan, J. A., and Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics) 28*, 1 (1979), 100–108.

[191] Hayashi, T., and Miyazaki, T. High-speed table lookup engine for IPv6 longest prefix match. In *IEEE global telecommunications conference* (1999), pp. 1576–1581.

[192] Hayes, J., and Danezis, G. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 1187–1203.

[193] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[194] He, Y., Faloutsos, M., Krishnamurthy, S., and Huffaker, B. On routing asymmetry in the internet. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.* (2005), vol. 2, IEEE, pp. 6–pp.

[195] He, Z., Zhang, T., and Lee, R. B. Machine learning based ddos attack detection from source side in cloud. In *The 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (2017), pp. 114–120.

[196] Henri, S., Garcia-Aviles, G., Serrano, P., Banchs, A., and Thiran, P. Protecting against website fingerprinting with multihoming. *Proceedings on Privacy Enhancing Technologies 2020*, 2 (2020), 89–110.

[197] Herrmann, D., Wendolsky, R., and Federrath, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), pp. 31–42.

[198] Hintz, A. Fingerprinting websites using traffic analysis. In *International workshop on privacy enhancing technologies* (2002), Springer, pp. 171–178.

[199] Hoang, N. P., Niaki, A. A., Gill, P., and Polychronakis, M. Domain name encryption is not enough: privacy leakage via ip-based website fingerprinting. *Proceedings on Privacy Enhancing Technologies 2021*, 4 (2021), 420–440.

[200] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[201] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials 16*, 4 (2014), 2037–2064.

[202] Hoi, S. C., Sahoo, D., Lu, J., and Zhao, P. Online learning: A comprehensive survey. *Neurocomputing 459* (2021), 249–289.

[203] Høiland-Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T., Ahern, D., and Miller, D. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies* (2018), pp. 54–66.

[204] Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., and Duan, H. How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018).

[205] Hornik, K., Stinchcombe, M., White, H., et al. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[206] Hossfeld, T., and Binzenhöfer, A. Analysis of skype voip traffic in umts: End-to-end qos and qoe measurements. *Computer Networks 52*, 3 (2008), 650–666.

[207] Howard, R. A. Dynamic programming and markov processes.

[208] Hu, X., Shu, Z., Song, X., Cheng, G., and Gong, J. Detecting cryptojacking traffic based on network behavior features. In *2021 IEEE Global Communications Conference (GLOBECOM)* (2021), IEEE, pp. 01–06.

[209] Huawei. Configuration guide - network management and monitoring. `https://support.huawei.com/enterprise/en/doc/EDOC1000178174/986bf11e/overview-of-netstream`. Accessed: 2022-02-11.

[210] i Muñoz, J. Z., Suárez-Varela, J., and Barlet-Ros, P. Detecting cryptocurrency miners with NetFlow/IPFIX network measurements. In *IEEE International Symposium on Measurements & Networking (M&N)* (2019), pp. 1–6.

[211] Ikawa, Y., Enoki, M., and Tatsubori, M. Location inference using microblog messages. In *Proceedings of the 21st international conference on world wide web* (2012), pp. 687–690.

[212] Iyer, S. G., and Pawar, A. D. Gpu and cpu accelerated mining of cryptocurrencies and their financial analysis. In *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on* (2018), IEEE, pp. 599–604.

[213] Jalili, R., Imani-Mehr, F., Amini, M., and Shahriari, H. R. Detection of distributed denial of service attacks using statistical pre-processor and unsupervised neural networks. In *International Conference on Information Security Practice and Experience* (2005), pp. 192–203.

[214] Janowitz, A. Statista: The statistics portal for market data, market research and market. `https://www.statista.com`, 2019.

[215] Jia, J., Wang, B., and Gong, N. Z. Random walk based fake account detection in online social networks. In *47th annual IEEE/IFIP international conference on dependable systems and networks* (2017), pp. 273–284.

[216] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia* (2014), pp. 675–678.

[217] Jiang, H., Moore, A. W., Ge, Z., Jin, S., and Wang, J. Lightweight application classification for network management. In *ACM SIGCOMM workshop on Internet network management* (2007).

[218] Jiang, M., Gou, G., Shi, J., and Xiong, G. I know what you are doing with remote desktop. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)* (2019), IEEE, pp. 1–7.

[219] Jmila, H., Blanc, G., Shahid, M. R., and Lazrag, M. A survey of smart home iot device classification using machine learning-based network traffic analysis. *IEEE Access 10* (2022), 97117–97141.

[220] Jordan, M. I., and Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science 349*, 6245 (2015), 255–260.

[221] Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 263–274.

[222] Juarez, M., Imani, M., Perry, M., Diaz, C., and Wright, M. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security* (2016), Springer, pp. 27–46.

[223] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research 4* (1996), 237–285.

[224] Kaminski, K. Mining for better threat intelligence: Cryptominer pools. `https://www.reliaquest.com/blog/mining-for-better-threat-intelligence-cryptominer-pools/`, 2020.

[225] Karagiannis, T., Papagiannaki, K., and Faloutsos, M. Blinc: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications* (2005), pp. 229–240.

[226] Karataş, A., and Şahin, S. A review on social bot detection techniques and research directions. In *International security and cryptology conference* (2017), pp. 156–161.

[227] Khirman, S., and Henriksen, P. Relationship between quality-of-service and quality-of-experience for public internet service. In *In Proc. of the 3rd Workshop on Passive and Active Measurement* (2002), vol. 1.

[228] Khokhar, M. J., Ehlinger, T., and Barakat, C. From network traffic measurements to qoe for internet video. In *2019 IFIP Networking Conference (IFIP Networking)* (2019), IEEE, pp. 1–9.

[229] Kim, H., Claffy, K. C., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference* (2008), pp. 1–12.

[230] Kokila, R., Selvi, S. T., and Govindarajan, K. Ddos detection and analysis in sdn-based environment using support vector machine classifier. In *The Sixth International Conference on Advanced Computing* (2014), IEEE, pp. 205–210.

[231] Koll, D., Li, J., Stein, J., and Fu, X. On the state of OSN-based Sybil defenses. In *IFIP networking* (2014), pp. 1–9.

[232] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 6 (2017), 84–90.

[233] Kumar, P. A. R., and Selvakumar, S. Distributed denial of service attack detection using an ensemble of neural classifier. *Computer Communications 34*, 11 (2011), 1328–1341.

[234] Lab, M. R. Monero, a private digital currency. `https://www.getmonero.org/`, 2018.

[235] Lab, M. R. Monero documentation - CryptoNight, a memory hard hash function. `https://monerodocs.org/proof-of-work/cryptonight/`, 2021.

[236] Lakhina, A., Papagiannaki, K., Crovella, M., Diot, C., Kolaczyk, E. D., and Taft, N. Structural analysis of network traffic flows. In *Proceedings of the joint international conference on Measurement and modeling of computer systems* (2004), pp. 61–72.

[237] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (2010), Hotnets-IX, pp. 19:1–19:6.

[238] Lashkari, A. H., Kadir, A. F. A., Gonzalez, H., Mbah, K. F., and Ghorbani, A. A. Towards a network-based framework for android malware detection and characterization. In *2017 15th Annual conference on privacy, security and trust (PST)* (2017), IEEE, pp. 233–23309.

[239] Laštovička, M., Špaček, S., Velan, P., and Čeleda, P. Using tls fingerprints for os identification in encrypted traffic. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium* (2020), IEEE, pp. 1–6.

[240] Le, A., Varmarken, J., Langhoff, S., Shuba, A., Gjoka, M., and Markopoulou, A. Antmonitor: A system for monitoring from mobile devices. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data* (2015), pp. 15–20.

[241] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[242] Lee, J. Calculating netflow volume. `https://www.plixer.com/blog/calculating-netflow-volume/`. Posted: 2013-03-26.

[243] Lee, K., Eoff, B. D., and Caverlee, J. Seven months with the devils: A long-term study of content polluters on Twitter. In *Fifth international AAAI conference on weblogs and social media* (2011).

[244] Lee, K., Kim, J., Kwon, K. H., Han, Y., and Kim, S. Ddos attack detection method using cluster analysis. *Expert systems with applications 34*, 3 (2008), 1659–1665.

[245] Lever, J. Classification evaluation: It is important to understand both what a classification metric expresses and what it hides. *Nature methods 13*, 8 (2016), 603–605.

[246] Li, J., Sisodia, D., and Stafford, S. On the detection of smart, self-propagating internet worms. *IEEE Transactions on Dependable and Secure Computing* (2022), 1–13.

[247] Li, J., and Stafford, S. Detecting smart, self-propagating internet worms. In *2014 IEEE Conference on Communications and Network Security* (2014), IEEE, pp. 193–201.

[248] Li, K., Zhou, W., Li, P., Hai, J., and Liu, J. Distinguishing DDoS attacks from flash crowds using probability metrics. In *2009 Third International Conference on Network and System Security* (2009), pp. 9–17.

[249] Li, P., Salour, M., and Su, X. A survey of internet worm detection and containment. *IEEE Communications Surveys & Tutorials 10*, 1 (2008), 20–35.

[250] Li, S., Guo, H., and Hopper, N. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 1977–1992.

[251] Li, X., Jiang, P., Chen, T., Luo, X., and Wen, Q. A survey on the security of blockchain systems. *Future Generation Computer Systems 107* (2020), 841–853.

[252] Liberatore, M., and Levine, B. N. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security* (2006), pp. 255–263.

[253] Limwiwatkul, L., and Rungsawang, A. Distributed denial of service detection using TCP/IP header and traffic measurement analysis. In *IEEE International Symposium on Communications and Information Technology* (2004), vol. 1, pp. 605–610.

[254] Lippmann, R., Fried, D., Piwowarski, K., and Streilein, W. Passive operating system identification from tcp/ip packet headers. In *Workshop on Data Mining for Computer Security* (2003), vol. 40.

[255] Liu, J., Fu, Y., Ming, J., Ren, Y., Sun, L., and Xiong, H. Effective and real-time in-app activity analysis in encrypted internet traffic streams. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (2017), pp. 335–344.

[256] Liu, X., Wang, W., Niyato, D., Zhao, N., and Wang, P. Evolutionary game for mining pool selection in blockchain networks. *IEEE Wireless Communications Letters 7*, 5 (2018), 760–763.

[257] Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., and Lloret, J. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE access 5* (2017), 18042–18050.

[258] Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., and Saberian, M. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing 24*, 3 (2020), 1999–2012.

[259] Lu, D., Bhat, S., Kwon, A., and Devadas, S. Dynaflow: An efficient website fingerprinting defense based on dynamically-adjusting flows. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society* (2018), pp. 109–113.

[260] Lu, K., Wu, D., Fan, J., Todorovic, S., and Nucci, A. Robust and efficient detection of ddos attacks for large-scale internet. *Computer Networks 51*, 18 (2007), 5036–5056.

[261] Lu, L., Chang, E.-C., and Chan, M. C. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security* (2010), Springer, pp. 199–214.

[262] Lundberg, S. M., and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems 30* (2017).

[263] Luo, X., Zhou, P., Chan, E. W., Lee, W., Chang, R. K., Perdisci, R., et al. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS* (2011), vol. 11.

[264] Massey Jr, F. J. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association 46*, 253 (1951), 68–78.

[265] Mazhar, M. H., and Shafiq, Z. Real-time video quality of experience monitoring for https and quic. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications* (2018), IEEE, pp. 1331–1339.

[266] McCoy, D., Bauer, K., Grunwald, D., Kohno, T., and Sicker, D. Shining light in dark places: Understanding the tor network. In *International symposium on privacy enhancing technologies symposium* (2008), Springer, pp. 63–76.

[267] McGregor, A., Hall, M., Lorier, P., and Brunskill, J. Flow clustering using machine learning techniques. In *International workshop on passive and active network measurement* (2004), Springer, pp. 205–214.

[268] McRobb, D. W. Cflowd design. *CAIDA, Sept* (1998).

[269] Meidan, Y., Bohadana, M., Shabtai, A., Guarnizo, J. D., Ochoa, M., Tippenhauer, N. O., and Elovici, Y. Profiliot: a machine learning approach for iot device identification based on network traffic analysis. In *Proceedings of the symposium on applied computing* (2017), pp. 506–509.

[270] Mergendahl, S., and Li, J. Rapid: Robust and adaptive detection of distributed denial-of-service traffic from the internet of things. In *2020 IEEE Conference on Communications and Network Security (CNS)* (2020), IEEE, pp. 1–9.

[271] mikeperry. Experimental defense for website traffic fingerprinting. `https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting/`, 2011. Accessed: 2022-02-11.

[272] Mirkovic, J., Feng, Y., and Li, J. Measuring changes in regional network traffic due to covid-19 stay-at-home measures. *arXiv preprint arXiv:2203.00742* (2022).

[273] Mirkovic, J., and Reiher, P. D-ward: a source-end defense against flooding denial-of-service attacks. *IEEE transactions on Dependable and Secure Computing 2*, 3 (2005), 216–232.

[274] Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Network and Distributed Systems Security (NDSS) Symposium* (2018).

[275] Mistry, S., and Raman, B. Quantifying traffic analysis of encrypted web-browsing., 1998. Project paper.

[276] Montgomery, D. C., Peck, E. A., and Vining, G. G. *Introduction to linear regression analysis*, vol. 821. 2012.

[277] Moore, A. W., and Zuev, D. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (2005), pp. 50–60.

[278] Mothilal, R. K., Sharma, A., and Tan, C. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency* (2020), pp. 607–617.

[279] Moustafa, R., and Slay, J. A comprehensive data set for network intrusion detection systems. *School of Engineering and Information Technology University of New South Wales at the Australian Defense Force Academy Canberra, Australia, UNSW-NB15* (2015).

[280] Moustis, D., and Kotzanikolaou, P. Evaluating security controls against HTTP-based DDoS attacks. In *IISA 2013*, pp. 1–6.

[281] Muja, M., and Lowe, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1) 2*, 331-340 (2009), 2.

[282] Murdoch, S. J., and Watson, R. N. Metrics for security and performance in low-latency anonymity systems. In *International Symposium on Privacy Enhancing Technologies Symposium* (2008), Springer, pp. 115–132.

[283] Murtagh, F., and Contreras, P. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2*, 1 (2012), 86–97.

[284] Naboulsi, D., Fiore, M., Ribot, S., and Stanica, R. Large-scale mobile traffic analysis: A survey. *IEEE Communications Surveys & Tutorials 18*, 1 (2016), 124–161.

[285] Najafabadi, M. M., Khoshgoftaar, T. M., Napolitano, A., and Wheelus, C. Rudy attack: Detection at the network level and its important features. In *The twenty-ninth international flairs conference* (2016).

[286] Nguyen, Q. P., Lim, K. W., Divakaran, D. M., Low, K. H., and Chan, M. C. Gee: A gradient-based explainable variational autoencoder for network anomaly detection. In *2019 IEEE Conference on Communications and Network Security (CNS)* (2019), IEEE, pp. 91–99.

[287] Nithyanand, R., Cai, X., and Johnson, R. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), pp. 131–134.

[288] Noble, W. S. What is a support vector machine? *Nature biotechnology 24*, 12 (2006), 1565–1567.

[289] Odintsov, P. FastNetMon–very fast DDoS analyzer with sflow/netflow/mirror support. `https://github.com/pavel-odintsov/fastnetmon/`.

[290] Oh, S. E., Sunkam, S., and Hopper, N. -fp: Extraction, classification, and prediction of website fingerprints with deep learning. *Proceedings on Privacy Enhancing Technologies 2019*, 3 (2019), 191–209.

[291] Orebaugh, A., Ramirez, G., and Beale, J. *Wireshark & Ethereal network protocol analyzer toolkit.* Elsevier, 2006.

[292] Orsini, C., King, A., Giordano, D., Giotsas, V., and Dainotti, A. BGPStream: a software framework for live and historical BGP data analysis. In *ACM Internet measurement conference* (2016), pp. 429–444.

[293] Orsolic, I., Pevec, D., Suznjevic, M., and Skorin-Kapov, L. Youtube qoe estimation based on the analysis of encrypted network traffic using machine learning. In *2016 IEEE Globecom Workshops (GC Wkshps)* (2016), IEEE, pp. 1–6.

[294] Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., and Aguilar, J. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials 21*, 2 (2018), 1988–2014.

[295] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website fingerprinting at internet scale. In *NDSS* (2016).

[296] Panchenko, A., Niessen, L., Zinnen, A., and Engel, T. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society* (2011), pp. 103–114.

[297] Papadogiannaki, E., Halevidis, C., Akritidis, P., and Koromilas, L. Otter: A scalable high-resolution encrypted traffic identification engine. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (2018), Springer, pp. 315–334.

[298] Papadogiannaki, E., and Ioannidis, S. Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware. *Sensors 21*, 4 (2021), 1140.

[299] Papadogiannaki, E., and Ioannidis, S. A survey on encrypted network traffic analysis applications, techniques, and countermeasures. *ACM Computing Surveys (CSUR) 54*, 6 (2021), 1–35.

[300] Park, K., and Willinger, W. Self-similar network traffic: An overview. *Self-Similar Network Traffic and Performance Evaluation* (2000), 1–38.

[301] Pastor, A., Mozo, A., Vakaruk, S., Canavese, D., López, D. R., Regano, L., Gómez-Canaval, S., and Lioy, A. Detection of encrypted cryptomining malware connections with machine and deep learning. *IEEE Access 8* (2020), 158036–158055.

[302] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (2019).

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

[304] Perry, M. Experimental defense for website traffic fingerprinting. `https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting/`. Posted: 2011-09-05.

[305] Petrov, I., Invernizzi, L., and Bursztein, E. Coinpolice: Detecting hidden cryptojacking attacks with neural networks. *arXiv preprint arXiv:2006.10861* (2020).

[306] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al. The design and implementation of open vswitch. In *The 12th NSDI* (2015), pp. 117–130.

[307] Phaal, P., Panchen, S., and McKee, N. InMon Corporation's sFlow: A method for monitoring traffic in switched and routed networks. RFC 3176, 2001.

[308] Pinheiro, A. J., Freitas de Araujo-Filho, P., de M. Bezerra, J., and Campelo, D. R. Adaptive packet padding approach for smart home networks: A tradeoff between privacy and performance. *IEEE Internet of Things Journal 8*, 5 (2021), 3930–3938.

[309] Prasad, K. M., Reddy, A. R. M., and Rao, K. V. Anomaly based real time prevention of under rated App-DDoS attacks on web: An experiential metrics based machine learning approach. *Indian Journal of Science and Technology* (2016).

[310] Praseed, A., and Thilagam, P. S. DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications. *IEEE Communications Surveys & Tutorials 21*, 1 (2018), 661–685.

[311] Quittek, J., Zseby, T., Claise, B., and Zander, S. Requirements for ip flow information export (ipfix). Tech. rep., RFC 3917 (informational), 2004.

[312] Rabiner, L., and Juang, B. An introduction to hidden markov models. *ieee assp magazine 3*, 1 (1986), 4–16.

[313] Rajahalme, J., Conta, A., Carpenter, B., and Deering, S. Ipv6 flow label specification. *RFC3697* (2004).

[314] Ranjan, S., Swaminathan, R., Uysal, M., and Knightly, E. W. DDoS-Resilient scheduling to counter application layer attacks under imperfect detection. In *INFOCOM* (2006).

[315] Rasley, J., Stephens, B., Dixon, C., Rozner, E., Felter, W., Agarwal, K., Carter, J., and Fonseca, R. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review 44*, 4 (2014), 407–418.

[316] Ratkiewicz, J., Conover, M. D., Meiss, M., Gonçalves, B., Flammini, A., and Menczer, F. M. Detecting and tracking political abuse in social media. In *Fifth international AAAI conference on weblogs and social media* (2011).

[317] Raymond, J.-F. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies* (2001), Springer, pp. 10–29.

[318] Razaghpanah, A., Vallina-Rodriguez, N., Sundaresan, S., Kreibich, C., Gill, P., Allman, M., and Paxson, V. Haystack: In situ mobile traffic analysis in user space. *arXiv preprint arXiv:1510.01419* (2015), 1–13.

[319] Recabarren, R., and Carbunar, B. Hardening stratum, the bitcoin pool mining protocol. *Proceedings on Privacy Enhancing Technologies*, 3 (2017), 57–74.

[320] Ren, J., Rao, A., Lindorfer, M., Legout, A., and Choffnes, D. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services* (2016), pp. 361–374.

[321] Rezaei, S., and Liu, X. How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets. *arXiv preprint arXiv:1812.09761* (2018).

[322] Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), pp. 1135–1144.

[323] Rimmer, V., Preuveneers, D., Juarez, M., Van Goethem, T., and Joosen, W. Automated website fingerprinting through deep learning.

[324] Rish, I., et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (2001), vol. 3, pp. 41–46.

[325] Rizzo, L. netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)* (2012), pp. 101–112.

[326] Roesslein, J. Tweepy documentation. `http://docs.tweepy.org/en/latest/`, 2009.

[327] Rosner, N., Kadron, I. B., Bang, L., and Bultan, T. Profit: Detecting and quantifying side channels in networked applications. In *NDSS* (2019).

[328] Rossi, D., and Valenti, S. Fine-grained traffic classification with NetFlow data. In *6th International wireless communications and mobile computing conference* (2010), pp. 479–483.

[329] Rossow, C. Amplification hell: Revisiting network protocols for ddos abuse. In *NDSS* (2014), pp. 1–15.

[330] Roughan, M., Sen, S., Spatscheck, O., and Duffield, N. Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification. In *4th ACM Internet measurement conference* (2004), pp. 135–148.

[331] RSnake, Kinsella, J., Gonzalez, H., and Lee, R. E. Slowloris HTTP DoS. `https://github.com/XCHADXFAQ77X/SLOWLORIS`, 2009.

[332] Ruffing, N., Zhu, Y., Libertini, R., Guan, Y., and Bettati, R. Smartphone reconnaissance: Operating system identification. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (2016), IEEE, pp. 1086–1091.

[333] Saltaformaggio, B., Choi, H., Johnson, K., Kwon, Y., Zhang, Q., Zhang, X., Xu, D., and Qian, J. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *10th {USENIX} Workshop on Offensive Technologies ({WOOT} 16)* (2016).

[334] Samek, W., and Müller, K.-R. Towards explainable artificial intelligence. In *Explainable AI: interpreting, explaining and visualizing deep learning.* Springer, 2019, pp. 5–22.

[335] Sang, A., and Li, S.-q. A predictability analysis of network traffic. *Computer networks 39*, 4 (2002), 329–345.

[336] Schneider, F., Feldmann, A., Krishnamurthy, B., and Willinger, W. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement* (2009), pp. 35–48.

[337] Schuster, R., Shmatikov, V., and Tromer, E. Beauty and the burst: Remote identification of encrypted video streams. In *26th {USENIX} Security Symposium ({USENIX} Security 17)* (2017), pp. 1357–1374.

[338] Seo, J., Lee, C., Shon, T., Cho, K.-H., and Moon, J. A new ddos detection model using multiple svms and tra. In *International Conference on Embedded and Ubiquitous Computing* (2005), Springer.

[339] Seufert, S., and O'Brien, D. Machine learning for automatic defence against distributed denial of service attacks. In *2007 IEEE International Conference on Communications* (2007), pp. 1217–1222.

[340] Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., and Elovici, Y. Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security 43* (2014), 1–18.

[341] Shalev-Shwartz, S., et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning 4*, 2 (2012), 107–194.

[342] Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q., and Xu, K. Machine learning-powered encrypted network traffic analysis: A comprehensive survey. *IEEE Communications Surveys & Tutorials* (2022), 1–1.

[343] Shen, M., Zhang, J., Zhu, L., Xu, K., Du, X., and Liu, Y. Encrypted traffic classification of decentralized applications on ethereum using feature fusion. In *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)* (2019), IEEE, pp. 1–10.

[344] Shiaeles, S. N., Katos, V., Karakos, A. S., and Papadopoulos, B. K. Real time ddos detection using fuzzy estimators. *computers & security 31*, 6 (2012), 782–790.

[345] Shone, N., Ngoc, T. N., Phai, V. D., and Shi, Q. A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence 2*, 1 (2018), 41–50.

[346] Shteiman, B. Hulk DoS tool. `https://github.com/grafov/hulk`, 2017.

[347] Simonovich, V. Imperva blocks our largest DDoS L7/brute force attack ever (peaking at 292,000 RPS). `https://www.imperva.com/blog/`, July 24, 2019.

[348] Sirinam, P., Imani, M., Juarez, M., and Wright, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 1928–1943.

[349] Sirinam, P., Mathews, N., Rahman, M. S., and Wright, M. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 1131–1148.

[350] Sivabalan, S., and Radcliffe, P. A novel framework to detect and block DDoS attack at the application layer. In *IEEE 2013 Tencon-Spring*, pp. 578–582.

[351] Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., and Sivaraman, V. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing 18*, 8 (2018), 1745–1759.

[352] Smith, J. M., and Schuchard, M. Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive bgp routing. In *2018 IEEE Symposium on Security and Privacy (S&P)* (2018), pp. 599–617.

[353] So-In, C. A survey of network traffic monitoring and analysis tools. *Cse 576m computer system analysis project, Washington University in St. Louis* (2009).

[354] Sommer, R., and Feldmann, A. NetFlow: Information loss or win? In *2nd ACM workshop on Internet measurment* (2002), pp. 173–174.

[355] Song, J., Lee, S., and Kim, J. Spam filtering in Twitter using sender-receiver relationship. In *International workshop on recent advances in intrusion detection* (2011), pp. 301–317.

[356] Song, Y., and Hengartner, U. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices* (2015), pp. 15–26.

[357] Storm Security. Application layer DDoS simulator. `https://stormsecurity.wordpress.com/2009/03/03/application-layer-ddos-simulator`.

[358] Stringhini, G., Kruegel, C., and Vigna, G. Detecting spammers on social networks. In *26th annual computer security applications conference* (2010), pp. 1–9.

[359] Sun, Q., Simon, D. R., Wang, Y.-M., Russell, W., Padmanabhan, V. N., and Qiu, L. Statistical identification of encrypted web browsing traffic. In *Proceedings 2002 IEEE Symposium on Security and Privacy* (2002), IEEE, pp. 19–30.

[360] Sunshine, C. A. *Computer network architectures and protocols.* Springer Science & Business Media, 2013.

[361] Suresh, M., and Anitha, R. Evaluating machine learning algorithms for detecting ddos attacks. In *Advances in Network Security and Applications* (Berlin, Heidelberg, 2011), D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds., Springer Berlin Heidelberg, pp. 441–452.

[362] Svoboda, J., Ghafir, I., Prenosil, V., et al. Network monitoring approaches: An overview. *Int J Adv Comput Netw Secur 5*, 2 (2015), 88–93.

[363] Tahaei, H., Afifi, F., Asemi, A., Zaki, F., and Anuar, N. B. The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications 154* (2020), 102538.

[364] Tahir, R., Durrani, S., Ahmed, F., Saeed, H., Zaffar, F., and Ilyas, S. The browsers strike back: countering cryptojacking and parasitic miners on the web. In *IEEE Conference on Computer Communications* (2019).

[365] Taleb, T., Fadlullah, Z. M., Hashimoto, K., Nemoto, Y., and Kato, N. Tracing back attacks against encrypted protocols. In *Proceedings of the 2007 international conference on Wireless communications and mobile computing* (2007), pp. 121–126.

[366] Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., and Ghogho, M. Deep learning approach for network intrusion detection in software defined networking. In *2016 international conference on wireless networks and mobile communications (WINCOM)* (2016), IEEE, pp. 258–263.

[367] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 439–454.

[368] Taylor, V. F., Spolaor, R., Conti, M., and Martinovic, I. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security 13*, 1 (2017), 63–78.

[369] Tegeler, F., Fu, X., Vigna, G., and Kruegel, C. BotFinder: Finding bots in network traffic without deep packet inspection. In *8th international conference on emerging networking experiments and technologies* (2012), pp. 349–360.

[370] Thomas, K., Grier, C., Ma, J., Paxson, V., and Song, D. Design and evaluation of a real-time URL spam filtering service. In *IEEE symposium on security and privacy* (2011), pp. 447–462.

[371] Thomas, K., Grier, C., and Paxson, V. Adapting social spam infrastructure for political censorship. In *5th USENIX workshop on large-scale exploits and emergent threats* (2012).

[372] Thomas, K., Grier, C., Song, D., and Paxson, V. Suspended accounts in retrospect: An analysis of Twitter spam. In *ACM Internet measurement conference* (2011), pp. 243–258.

[373] Thomas, R., Mark, B., Johnson, T., and Croall, J. Netbouncer: client-legitimacy-based high-performance ddos filtering. In *Proceedings DARPA Information Survivability Conference and Exposition* (2003), vol. 1, IEEE, pp. 14–25.

[374] Tjoa, E., and Guan, C. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE transactions on neural networks and learning systems 32*, 11 (2020), 4793–4813.

[375] Trammell, B., and Boschi, E. An introduction to ip flow information export (ipfix). *IEEE Communications Magazine 49*, 4 (2011), 89–95.

[376] Trestian, I., Ranjan, S., Kuzmanovic, A., and Nucci, A. Measuring serendipity: connecting people, locations and interests in a mobile 3g network. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement* (2009), pp. 267–279.

[377] Unit 42. Highlights from the unit 42 cloud threat report, 1h 2021. `https://unit42.paloaltonetworks.com/highlights-cloud-threat-report-1h-2021/`, 2021.

[378] Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.-L. A., Elkhatib, Y., Hussain, A., and Al-Fuqaha, A. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE access 7* (2019), 65579–65615.

[379] Van Der Maaten, L., Postma, E., Van den Herik, J., et al. Dimensionality reduction: a comparative. *J Mach Learn Res 10*, 66-71 (2009), 13.

[380] van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D. J., Lindorfer, M., Choffnes, D., van Steen, M., and Peter, A. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and Distributed System Security Symposium (NDSS)* (2020), vol. 27.

[381] Varlioglu, S., Gonen, B., Ozer, M., and Bastug, M. F. Is cryptojacking dead after coinhive shutdown? *arXiv preprint arXiv:2001.02975* (2020).

[382] Velan, P., Čermák, M., Čeleda, P., and Drašar, M. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management 25*, 5 (2015), 355–374.

[383] Voigt, P., and Von dem Bussche, A. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing 10*, 3152676 (2017), 10–5555.

[384] Wagner, T. D., Mahbub, K., Palomar, E., and Abdallah, A. E. Cyber threat intelligence sharing: Survey and research directions. *Computers & Security 87* (2019), 101589.

[385] Waldbusser, S., Cole, R., Kalbfleisch, C., and Romascanu, D. Introduction to the remote monitoring (rmon) family of mib modules. *RFC3577, Network Working Group* (2003).

[386] WALLACE, R. S. *Be Your Own Botmaster: The Step By Step Guide to Creating, Hosting and Selling Your Own AI Chat Bot On Pandorabots.* ALICE AI foundations, Incorporated, 2003.

[387] WANG, A. H. Don't follow me: Spam detection in Twitter. In *International conference on security and cryptography* (2010), pp. 1–10.

[388] WANG, B., GONG, N. Z., AND FU, H. GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In *IEEE international conference on data mining* (2017), pp. 465–474.

[389] WANG, B., JIA, J., AND GONG, N. Z. Graph-based security and privacy analytics via collective classification with joint weight learning and propagation. In *The network and distributed system security symposium* (2019).

[390] WANG, B., ZHANG, L., AND GONG, N. Z. SybilSCAR: Sybil detection in online social networks via local rule based propagation. In *IEEE conference on computer communications* (2017), pp. 1–9.

[391] WANG, C., DANI, J., LI, X., JIA, X., AND WANG, B. Adaptive fingerprinting: website fingerprinting over few encrypted traffic. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy* (2021), pp. 149–160.

[392] WANG, G., KONOLIGE, T., WILSON, C., WANG, X., ZHENG, H., AND ZHAO, B. Y. You are how you click: Clickstream analysis for Sybil detection. In *USENIX Security* (2013), pp. 241–256.

[393] WANG, G., MOHANLAL, M., WILSON, C., WANG, X., METZGER, M., ZHENG, H., AND ZHAO, B. Y. Social turing tests: Crowdsourcing Sybil detection. In *The network and distributed system security symposium* (2013).

[394] WANG, G., WANG, T., ZHENG, H., AND ZHAO, B. Y. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *USENIX security symposium* (2014), pp. 239–254.

[395] WANG, H., XU, F., LI, Y., ZHANG, P., AND JIN, D. Understanding mobile traffic patterns of large scale cellular towers in urban environment. In *Proceedings of the 2015 Internet Measurement Conference* (2015), pp. 225–238.

[396] WANG, J., PHAN, R. C.-W., WHITLEY, J. N., AND PARISH, D. J. Augmented attack tree modeling of distributed denial of services and tree based attack detection method. In *The 10th IEEE International Conference on Computer and Information Technology* (2010), pp. 1009–1014.

[397] Wang, L.-M., Miskell, T., Morgan, J., and Verplanke, E. Design of a real-time traffic mirroring system. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2021), IEEE, pp. 793–796.

[398] Wang, N., Ho, K. H., Pavlou, G., and Howarth, M. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials 10*, 1 (2008), 36–56.

[399] Wang, Q., Yahyavi, A., Kemme, B., and He, W. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *2015 IEEE conference on communications and network security (CNS)* (2015), IEEE, pp. 433–441.

[400] Wang, S., Chen, Z., Zhang, L., Yan, Q., Yang, B., Peng, L., and Jia, Z. Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)* (2016), IEEE, pp. 1–6.

[401] Wang, T. The one-page setting: A higher standard for evaluating website fingerprinting defenses. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), pp. 2794–2806.

[402] Wang, T., Cai, X., Nithyanand, R., Johnson, R., and Goldberg, I. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 143–157.

[403] Wang, T., and Goldberg, I. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), pp. 201–212.

[404] Wang, T., and Goldberg, I. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *26th {USENIX} Security Symposium ({USENIX} Security 17)* (2017), pp. 1375–1390.

[405] Wang, W., Ferrell, B., Xu, X., Hamlen, K. W., and Hao, S. Seismic: Secure in-lined script monitors for interrupting cryptojacks. In *European Symposium on Research in Computer Security* (2018), Springer, pp. 122–142.

[406] Wang, W., Shang, Y., He, Y., Li, Y., and Liu, J. Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. *Information Sciences 511* (2020), 284–296.

[407] Wang, X., and Yin, M. Are explanations helpful? a comparative study of the effects of explanations in ai-assisted decision-making. In *26th International Conference on Intelligent User Interfaces* (2021), pp. 318–328.

[408] Wang, Y., Li, Z., Gou, G., Xiong, G., Wang, C., and Li, Z. Identifying dapps and user behaviors on ethereum via encrypted traffic. In *International Conference on Security and Privacy in Communication Systems* (2020), Springer, pp. 62–83.

[409] Wang, Y., Zheng, N., Xu, M., Qiao, T., Zhang, Q., Yan, F., and Xu, J. Hierarchical identifier: Application to user privacy eavesdropping on mobile payment app. *Sensors 19*, 14 (2019), 3052.

[410] Watkins, C. J., and Dayan, P. Q-learning. *Machine learning 8*, 3-4 (1992), 279–292.

[411] Wei, X., Gomez, L., Neamtiu, I., and Faloutsos, M. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking* (2012), pp. 137–148.

[412] Wichtlhuber, M., Strehle, E., Kopp, D., Prepens, L., Stegmueller, S., Rubina, A., Dietzel, C., and Hohlfeld, O. Ixp scrubber: learning from blackholing traffic for ml-driven ddos detection at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 707–722.

[413] Wikipedia. Argus – audit record generation and utilization system. `https://en.wikipedia.org/wiki/Argus_%E2%80%93_Audit_Record_Generation_and_Utilization_System`. Accessed: 2022-02-11.

[414] Wikipedia. Internet protocol suite. `https://en.wikipedia.org/wiki/Internet_protocol_suite`. Date of visit: 2021-10-05.

[415] Wold, S., Esbensen, K., and Geladi, P. Principal component analysis. *Chemometrics and intelligent laboratory systems 2*, 1-3 (1987), 37–52.

[416] Wolff, E. D., Growley, K., Gruden, M., et al. Navigating the solarwinds supply chain attack. *The Procurement Lawyer 56*, 2 (2021).

[417] Wright, C. V., Ballard, L., Coull, S. E., Monrose, F., and Masson, G. M. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *2008 IEEE Symposium on Security and Privacy (sp 2008)* (2008), IEEE, pp. 35–49.

[418] Wright, C. V., Coull, S. E., and Monrose, F. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS* (2009), vol. 9, Citeseer.

[419] Wu, T., Breitinger, F., and Niemann, S. Iot network traffic analysis: Opportunities and challenges for forensic investigators? *Forensic Science International: Digital Investigation 38* (2021), 301123.

[420] Xie, Y., and Yu, S. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking 17*, 1 (Feb 2009), 15–25.

[421] Xie, Y., and Yu, S.-Z. A novel model for detecting application layer DDoS attacks. In *First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)* (2006), vol. 2, pp. 56–63.

[422] Xu, C., Zhao, G., Xie, G., and Yu, S. Detection on application layer DDoS using random walk model. In *2014 IEEE International Conference on Communications (ICC)*, pp. 707–712.

[423] Xu, F., Li, Y., Wang, H., Zhang, P., and Jin, D. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM transactions on networking 25*, 2 (2016), 1147–1161.

[424] Xu, F., Lin, Y., Huang, J., Wu, D., Shi, H., Song, J., and Li, Y. Big data driven mobile traffic understanding and forecasting: A time series approach. *IEEE transactions on services computing 9*, 5 (2016), 796–805.

[425] Xu, J., Fan, J., Ammar, M., and Moon, S. B. On the design and performance of prefix-preserving ip traffic trace anonymization. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), pp. 263–266.

[426] Xu, S., Sen, S., and Mao, Z. M. Csi: Inferring mobile abr video adaptation behavior under https and quic. In *Proceedings of the Fifteenth European Conference on Computer Systems* (2020), pp. 1–16.

[427] Yadav, S., and Subramanian, S. Detection of application layer DDoS attack by feature learning using stacked autoencoder. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)* (2016), pp. 361–366.

[428] Yan, F., Xu, M., Qiao, T., Wu, T., Yang, X., Zheng, N., and Choo, K.-K. R. Identifying wechat red packets and fund transfers via analyzing encrypted network traffic. In *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)* (2018), IEEE, pp. 1426–1432.

[429] Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B. Y., and Dai, Y. Uncovering social network Sybils in the wild. *ACM Transactions on Knowledge Discovery from Data 8*, 1 (2014), 1–29.

[430] Yao, H., Gao, P., Wang, J., Zhang, P., Jiang, C., and Han, Z. Capsule network assisted iot traffic classification mechanism for smart cities. *IEEE Internet of Things Journal 6*, 5 (2019), 7515–7525.

[431] Yardi, S., Romero, D., Schoenebeck, G., et al. Detecting spam in a Twitter network. *First Monday* (2010).

[432] Yatagai, T., Isohara, T., and Sasase, I. Detection of HTTP-GET flood attack based on analysis of page access behavior. In *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 232–235.

[433] Yin, Q., Liu, Z., Li, Q., Wang, T., Wang, Q., Shen, C., and Xu, Y. Automated multi-tab website fingerprinting attack. *IEEE Transactions on Dependable and Secure Computing* (2021).

[434] Yu, S.-Z., and Xie, Y. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM transactions on networking 17*, 1 (2009), 54–65.

[435] Yuan, X., Li, C., and Li, X. Deepdefense: identifying ddos attack via deep learning. In *IEEE International Conference on Smart Computing* (2017), pp. 1–8.

[436] Zekri, M., El Kafhali, S., Aboutabit, N., and Saadi, Y. Ddos attack detection using machine learning techniques in cloud computing environments. In *The 3rd International Conference of Cloud Computing Technologies and Applications* (2017), IEEE, pp. 1–7.

[437] ZeroDot1. Simple lists that can help prevent cryptomining in the browser or other applications. `https://gitlab.com/ZeroDot1/CoinBlockerLists`, 2020.

[438] Zhang, G., Jiang, S., Wei, G., and Guan, Q. A prediction-based detection algorithm against distributed denial-of-service attacks. In *International conference on wireless communications and mobile computing: Connecting the World wirelessly* (2009), pp. 106–110.

[439] Zhang, H., Taha, A., Trapero, R., Luna, J., and Suri, N. Sentry: A novel approach for mitigating application layer DDoS threats. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 465–472.

[440] Zhang, J., Zhang, R., Zhang, Y., and Yan, G. The rise of social botnets: Attacks and countermeasures. *IEEE Transactions on Dependable and Secure Computing 15*, 6 (2016), 1068–1082.

[441] Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., and Wu, J. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)* (2020).

[442] Zhang, M.-L., and Zhou, Z.-H. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition 40*, 7 (2007), 2038–2048.

[443] Zhang, S., Wang, Z., Yang, J., Cheng, X., Ma, X., Zhang, H., Wang, B., Li, Z., and Wu, J. Minehunter: A practical cryptomining traffic detection algorithm based on time series tracking. In *Annual Computer Security Applications Conference* (2021), pp. 1051–1063.

[444] Zhao, Y., Ma, X., Li, J., Yu, S., and Li, W. Revisiting website fingerprinting attacks in real-world scenarios: A case study of shadowsocks. In *International Conference on Network and System Security* (2018), Springer, pp. 319–336.

[445] Zhou, Y., Sun, C., Liu, H. H., Miao, R., Bai, S., Li, B., Zheng, Z., Zhu, L., Shen, Z., Xi, Y., et al. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 76–89.

[446] Zi, L., Yearwood, J., and Wu, X.-W. Adaptive clustering with feature ranking for ddos attacks detection. In *The Fourth International Conference on Network and System Security* (2010), IEEE, pp. 281–286.

[447] Zimba, A., Wang, Z., Mulenga, M., and Odongo, N. H. Crypto mining attacks in information systems: An emerging threat to cyber security. *Journal of Computer Information Systems* (2018), 1–12.