Enterprise Network Security via Data-driven Methods and Programmable Network Telemetry

Minzhao Lyu

A dissertation submitted in fulfillment
of the requirements for the degree of

Doctor of Philosophy

School of Electrical Engineering and Telecommunications

The University of New South Wales

September 2021

Thesis submission for the degree of Doctor of Philosophy

Thesis Title and Abstract | Declarations | Inclusion of Publications Statement | Corrected Thesis and Responses

Thesis Title

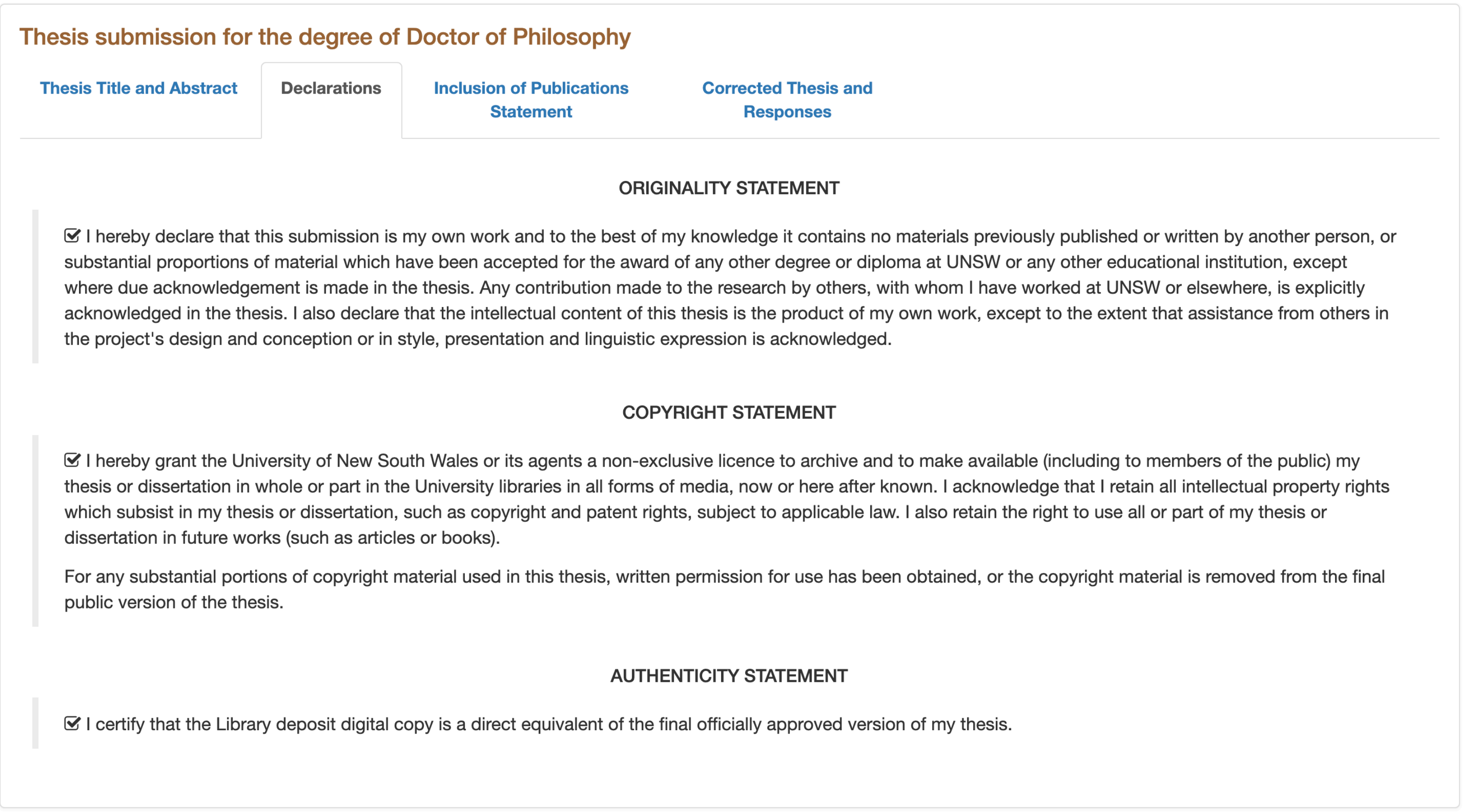
Enterprise Network Security via Data-driven Methods and Programmable Network Telemetry

Thesis Abstract

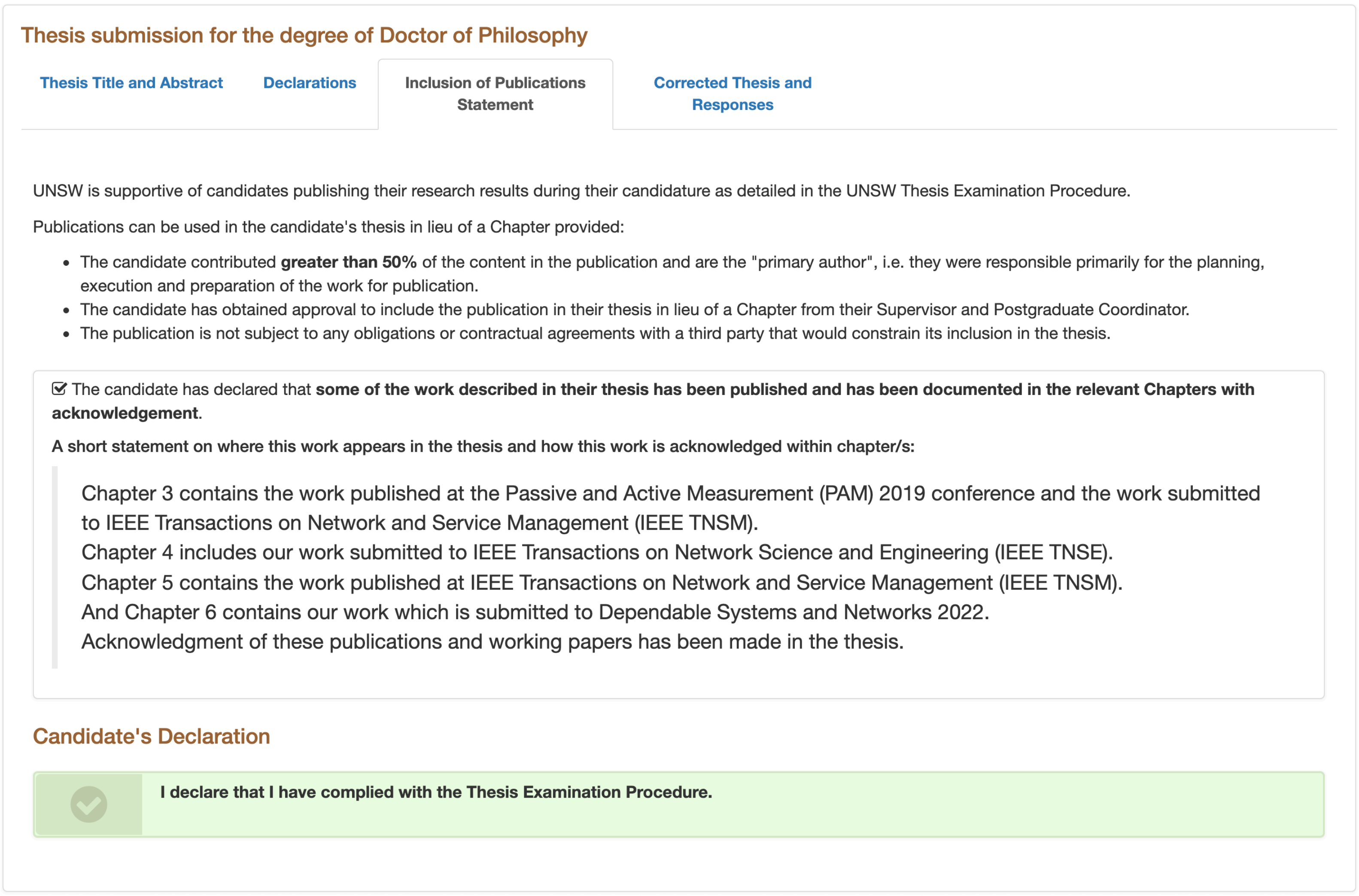
Enterprise networks are both complex and dynamic, with various kinds of servers, clients, and cyber-physical devices being deployed and removed. It is not surprising that IT departments struggle to track their connected assets, monitor their operational health, understand the attack surface, and protect them from cyber threats. Current security systems are unable to cope with the growth of emerging cyber threats. Hardware solutions are expensive and inflexible as their high-speed performance is optimized for static rulesets. Software solutions have great flexibility, but struggle to cope with high data rates which limit their telemetry granularity for embedded threats. Our thesis proposes approaches that combine hardware performance with software flexibility, by leveraging Programmable Networks (PN) and Machine Learning (ML). Telemetry from Terabit-speed PN Switches is used to extract network attributes, and this is combined with ML models of asset behavior to monitor their health and to detect attacks. We make four key contributions.

Our first contribution focuses on the Domain Name System (DNS). We develop a clustering method to classify DNS assets and track their health through well-articulated monitoring metrics. We demonstrate that our method successfully identifies key DNS assets and is further able to make recommendations on how these assets can be better secured against misuse. The second contribution extends our asset classification beyond DNS. We develop a system to extract telemetry, feeds the attributes to a multi-grained ML scheme that classifies the assets in real-time, and reactively collects packet telemetry of suspicious hosts for forensics analysis. The third contribution detects DNS-based network attacks on enterprise hosts. We develop a hierarchical anomaly detection method that profiles incoming DNS traffic at various levels of hierarchy to isolate DNS attackers that could be stealthy and distributed. Our fourth contribution expands the attack detection to the entire network-level. We develop a progressive inference architecture to detect attacks through a series of stages each with increasing telemetry cost but narrowing focus.

Taken together, our contributions apply PN and ML to develop practical and effective ways that give enterprise IT departments continuous visibility of their assets, advance warning of the threat surface, and real-time alarms when network attacks unfold.


Thesis Title and Abstract
Declarations
Inclusion of Publications Statement
Corrected Thesis and Responses

Thesis submission for the degree of Doctor of Philosophy



Thesis submission for the degree of Doctor of Philosophy

Thesis Title and Abstract
Declarations
Inclusion of Publications Statement
Corrected Thesis and Responses

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in the candidate's thesis in lieu of a Chapter provided:

- The candidate contributed **greater than 50%** of the content in the publication and are the "primary author", i.e. they were responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has obtained approval to include the publication in their thesis in lieu of a Chapter from their Supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis.

✓ The candidate has declared that **some of the work described in their thesis has been published and has been documented in the relevant Chapters with acknowledgement**.

A short statement on where this work appears in the thesis and how this work is acknowledged within chapter/s:

> Chapter 3 contains the work published at the Passive and Active Measurement (PAM) 2019 conference and the work submitted to IEEE Transactions on Network and Service Management (IEEE TNSM).
> Chapter 4 includes our work submitted to IEEE Transactions on Network Science and Engineering (IEEE TNSE).
> Chapter 5 contains the work published at IEEE Transactions on Network and Service Management (IEEE TNSM).
> And Chapter 6 contains our work which is submitted to Dependable Systems and Networks 2022.
> Acknowledgment of these publications and working papers has been made in the thesis.

Candidate's Declaration

✓ **I declare that I have complied with the Thesis Examination Procedure.**

Abstract

Enterprise networks are both complex and dynamic, with various kinds of servers (web, email, VPN, storage), clients (fixed, wireless), and Internet-of-Things devices (cameras, printers, sensors) being deployed, moved, and removed continuously. Furthermore, these assets are spread across various network segments (*e.g.,* VLANs), often managed by different departments, with complex interconnection rules between segments, to public/private cloud services, and to the general Internet. It is therefore not surprising that organizational IT departments struggle to track their connected assets, monitor their operational health, understand the attack surface they expose, and protect them from external as well as internal threats.

Current enterprise security systems such as Next-Generation-Firewalls (NGFW) and intrusion detection systems (IDS) are unable to cope with the growing volumes and diversity of emerging cyber-threats. Hardware appliance based solutions are not just expensive, but also inflexible as their high-speed performance is optimized for relatively static rulesets. Software solutions on the other hand have great flexibility, but struggle to cope with high data rates which limit the granularity at which they analyze traffic for embedded threats. To advance the state-of-the-art of enterprise asset monitoring and distributed network attack detection, my thesis proposes a new approach that combines hardware performance with software flexibility, by leveraging the concepts of Programmable Network (PN) and Machine Learning (ML). Telemetry from Terabit-speed Programmable Switches is used to extract key attributes of traffic streams, and this is combined with ML models of enterprise asset behavior to monitor their health and to detect attacks. I make four key contributions.

My **first contribution** focuses on the Domain Name System (DNS). I analyze DNS traffic from two large organizations to identify the behavioral aspects of various DNS assets. Using the behavioral attributes, I develop a clustering method to classify assets (*e.g.,* recursive resolvers and authoritative name servers) and track their health through a set of well-articulated monitoring metrics. I demonstrate that my method successfully identifies over 100 key DNS assets in the two organizations and is further able to make recommendations on how these assets can be better secured against misuse.

The **second contribution** extends my enterprise asset classification beyond DNS to include other asset types such as web servers, VPN servers, and file storage servers. For this, I develop a system that uses Programmable Network techniques to extract telemetry efficiently, feeds the attributes to a multi-grained ML-based scheme that classifies the assets in real-time, and reactively collects packet-level telemetry of suspicious hosts for forensics analysis. My method identifies hundreds of typical servers and thousands of less common assets (*e.g.,* LDAP server and Redis proxy) across the two organizations. It additionally highlights instances of atypical behavior that provide advance warnings to IT staff on potentially anomalous assets.

The **third contribution** detects DNS-based network attacks on enterprise hosts. To this end, I analyze incoming DNS traffic to the two organizations, and develop a hierarchical anomaly detection method that profiles incoming DNS traffic at various levels of hierarchy (*e.g.,* host, subnet, and AS) to isolate DNS attackers that could be stealthy and distributed. The models I train detect DNS attacks in lab data with over 99% accuracy at each level of the hierarchy, and in a 1-month trial in the wild reveal hundreds of attacks that were missed by the organizational firewalls.

My **fourth contribution** expands the attack detection from DNS to the whole dimension of network traffic. To achieve both detection effectiveness and operational practicality, I develop a multi-stage progressive inference architecture to optimally detect network attacks through a series of stages (*e.g.,* active enterprise hosts, victims, distributed attackers, and malicious flows) each with increasing telemetry cost but narrowing focus. Evaluations using real distributed denial-of-service (DDoS) attacks and large-scale enterprise traffic traces demonstrate the ability of my system in detecting distributed network attacks to the finest flow-level with practically low computational costs as around 30% CPU and 8% RAM usage on a typical blade server, which is not achievable by its counterpart solutions.

Taken together, my contributions apply Programmable Network and Machine Learning to develop new practical and effective ways that give enterprise IT departments continuous visibility of their assets, advance warning of the threat surface they expose, and real-time alarms when network attacks unfold.

List of Publications

During the course of this thesis project, a number of publications have been made based on the work presented here and are listed below for reference.

Journal Publications

1. **M. Lyu**, H. Habibi Gharakheili, C. Russell and V. Sivaraman, “Analyzing Enterprise DNS Traffic to Classify Assets and Track Cyber-Health”, **under review** at *IEEE Transactions on Network and Service Management.*

2. **M. Lyu**, H. Habibi Gharakheili, and V. Sivaraman, “Classifying and Tracking Enterprise Assets via Network Behavioral Analysis”, **under review** at *IEEE Transactions on Network Science and Engineering.*

3. **M. Lyu**, H. Habibi Gharakheili, C. Russell and V. Sivaraman, “Hierarchical Anomaly-Based Detection of Distributed DNS Attacks on Enterprise Networks”, *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 1031-1048, March 2021.

Conference Publications

5. **M. Lyu**, H. Habibi Gharakheili, and V. Sivaraman, “PEDDA: Practical and Effective Detection of Distributed Attacks on Enterprise Networks via Multi-stage Progressive Inference”, **under review** at *IEEE/IFIP International Conference on Dependable Systems and Networks*, Baltimore, Maryland, USA, June 2022.

6. **M. Lyu**, H. Habibi Gharakheili, C. Russell and V. Sivaraman, “Mapping an Enterprise Network by Analyzing DNS Traffic”, *Passive and Active Measurement Conference*, Puerto Varas, Chile, March 2019.

In addition, a number of publications have been made based on my research collaborations on other projects and are listed below for reference.

Miscellaneous Publications

7. B. Chen, S. Qiao, J. Zhao, D. Liu, X. Shi, **M. Lyu**, H. Chen, H. Lu and Y. Zhai,"A Security Awareness and Protection System for 5G Smart Healthcare Based on Zero-Trust Architecture", *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10248-10263, 1 July1, 2021.

8. F. Cui, X. He, Y. Zhai, **M. Lyu**, J. Shi, D. Sun, S. Jiang, C. Li and J. Zhao,"Application of Telemedicine Services Based on a Regional Telemedicine Platform in China From 2014 to 2020: Longitudinal Trend Analysis", *Journal of Medical Internet Research*, vol. 23, no. 7, July 2021.

9. H. Habibi Gharakheili, **M. Lyu**, Y. Wang, H. Kumar and V. Sivaraman, "iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification", *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1071-1085, Sept. 2019.

10. S. C. Madanapalli, **M. Lyu**, H. Kumar, H. Habibi Gharakheili and V. Sivaraman, "Real-Time Detection, Isolation and Nonitoring of Elephant Flows using Commodity SDN System", *IEEE/IFIP Network Operations and Management Symposium*, April 2018.

Acknowledgment

Pursuing a Ph.D. degree is a tough journey, especially under the heavy impact of COVID-19. However, with the warm and inspirational supports from my supervisors, colleagues, friends, and families, the journey becomes enjoyable and rewarding.

I first and foremost thank my supervisors at both UNSW (Prof. Vijay Sivaraman and Dr. Hassan Habibi Gharakheili) and CSIRO's Data61 (Dr. Craig Russell), who have become extremely helpful and supportive not only for my research projects but also for my livelong successes. I would like to express my sincere gratitude to Prof. Vijay Sivaraman. I am deeply impressed by Vijay's motivational leadership, sharp mind, critical thinking, and accurate perception of the research field as a great scientist. Without his professional guidance, constructive feedback, and patient help, the completion of this thesis and my future success would definitely become impossible. I would give my ultimate gratitude to Dr. Hassan Habibi Gharakheili. Hassan has consistently provided me enlightening suggestions, detailed comments, and close collaborations that pave the path for my Ph.D. journey. I would give my huge gratitude to Dr. Craig Russell for his support and help during my Ph.D. studies.

I would like to thank the University of New South Wales (UNSW), and Commonwealth Scientific and Industrial Research Organization (CSIRO) – Data61 for their accepting me as a Ph.D. candidate and providing financial supports, academic resources, laboratory equipment, and a comfortable office environment so that I could conduct my research studies with minimum difficulty.

Affected by the COVID-19 travel restrictions, I was not able to come back to Sydney from my home city after my annual break in 2020. Thankfully, the National Telemedicine Center of China (NTCC), the First Affiliated Hospital of Zhengzhou University offered me a great research internship position. Under the pleasant academic atmosphere, convenient office facility, and robust Internet connectivity, I could continue my Ph.D. research remotely and have regular online discussions with my university supervisors. Moreover, I have been offered the opportunity to participate in cutting-edge research projects in network security and telemedicine systems for the “largest hospital in the world”. I would like to express my great appreciation to Prof. Jie Zhao for providing this amazing opportunity. I would also like to thank my leaders, collaborators, colleagues, and friends in NTCC: Prof. Yunkai Zhai, Baozhan Chen, Xianying He, Dr. Dongqing Liu, Xiaobing Shi, Shansong Huang, Wenchao Wang, Yanjia Zhang, Guoshu Huang, Fangfang Cui, Jinming Shi, Dongxu Sun, Haotian Chen, Dr. Shuai Jiang, Chenchen Li, Dr. Siyuan Qiao, and Dr. Huimin Lu. Thank you all for giving me such a fantastic internship experience.

My sincere thanks to Dr. Dinusha Vatsalan at CSIRO’s Data61, Belinda Lee at UNSW GRS, Vanessa Chen at UNSW Student Centre, and many other people who have provided their kind supports during my application of COVID-19 travel exemption to enter Australia and my 14-day hotel quarantine. Without your help, it would not be possible for me to continue my on-site research under such a difficult time.

I would like to thank my colleagues and friends Arman, Sharat, Jawad, Ayyoob, Tara, Arunan, Irsha, Yu, Ke, Himal, Qasim, and many others who have provided encouragement and happiness throughout my Ph.D. journey.

Pursuing the Ph.D. is a long journey that I meet too many people deserving my thank and gratitude. Most importantly, I would highlight that without the selfless support and eternal love from my parents, in-laws, wife, and families, it would be not possible for me to reach this important milestone of my life. No language and word can express my deep gratitude to them.

Contents

List of Figures

List of Tables

Acronyms

3GPP	3rd Generation Partnership Project
AI	Artifical Intelligence
ALU	Arithmetic Logic Unit
API	Application Programming Interface
AS	Autonomous System
BYOD	Bring Your Own Device
BYOT	Bring Your Own Technology
CCDF	Complementary Cumulative Distribution Function
CPU	Central Processing Unit
DDoS	Distributed Denial-of-Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DoS	Denial-of-Service
DPDK	Data Plane Development Kit
EM	Expectation-Maximization
FTP	File Transfer Protocol
Gbps	Gigabits per second
HC	Hierarchical Clustering
HNB	Hidden Naive Bayes
HTTP	Hypertext Transfer Protocol

HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	Identifier
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
IT	Information Technology
LDAP	Lightweight Directory Access Protocol
Mbps	Megabits per second
ML	Machine Learning
MLP	Multilayer Perceptron
NAT	Network Address Translation
NFV	Network Function Virtualization
NGFW	Next Generation Firewall
NTP	Network Time Protocol
P2P	Peer-to-Peer
PC	Personal Computer
PCA	Principal Component Analysis
PCAP	Packet Capture
PN	Programmable Networks
PPS	Packets Per Second

QUIC	Quick UDP Internet Connection
RAM	Random Access Memory
RBF	Radial Basis Function
RF	Random Forest
SDN	Software-Defined Networking
SFTP	Secure File Transfer Protocol
SIEM	Security Information and Event Management
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SRAM	Static Random-Access Memory
SSDP	Simple Service Discovery Protocol
SSE	Sum of Squared Errors
SSH	Secure Shell
SSL	Secure Sockets Layer
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VNF	Virtual Network Function
VPN	Virtual Private Network

Chapter 1

Introduction

Contents

In recent years, I have witnessed the revolutionary advances of communication and network technologies that provide unprecedented connectivity for human societies, such as high-speed low-latency broadband networks, the fifth generation of mobile networks (5G), cyber-physical systems, Internet-of-Things (IoT) technology, and edge computing. Along with the rapid development of network technologies, cyber-attacks have inevitably become sophisticated, agile, destructive, and stealthy as malicious actors now have access to a huge pool of resources they can exploit such as vulnerable communication protocols, insecure devices, and even networked infrastructures with improper configurations in critical industries like electrical grids. Therefore, enforcing network security has become a serious task for both industrial and academic communities.

Enterprises such as universities, research institutes, banks, and hospitals have complex network environments as they are operating a large variety of critical network assets including websites, authoritative name servers, databases, virtual private network (VPN) servers, application proxies, WiFi access points, IoT appliances, and others. Staff and visitors of an enterprise also connect their personal Bring-Your-Own-Technology (BYOT) devices like mobile phones and laptops to enterprise networks through wired or wireless access gateways. Besides, servers inside the networks may also archive highly valuable and sensitive data such as user credentials, research manuscripts, financial records, and patient's bio-information. Thus, the highly capable assets, complex compositions, and valuable data of an enterprise network unavoidably make it an attractive target of network attacks. For example, servers and network infrastructures that provide services via different protocols and platforms may expose security vulnerabilities of various kinds to malicious actors on the public Internet. As a consequence, they are popular targets of network attacks such as reconnaissance scans that discover host vulnerabilities and distributed Denial-of-Service (DDoS) attacks to paralyze certain applications or exhaust network resources of the victim asset. In addition to attacks from the external side, internal assets and end-host devices (*e.g.,* IoTs, PCs, tablets, and mobile phones) might also be misused or infected by malware to conduct malicious network activities such as generating/reflecting attacks, spreading computer virus inside the network, and exfiltrating sensitive data to external attackers.

To ensure the correct operation of networked assets within an enterprise and protect them from potential network attacks, IT departments are expected to continuously monitor the behavior of enterprise hosts and effectively detect network attacks of dynamic patterns and complex forms. For asset network behavioral monitoring, IT departments nowadays are primarily utilizing static configurations (*e.g.,* DHCP records, DNS maps, or firewall policies) on their management systems which are hard to keep updated with the ever-increasing complexity of host functionalities and their fast-changing communication patterns. There are also many dynamic solutions developed by the research community that leverage real-time network telemetry and flow statistics to track traffic between networked hosts. However, such fine-grained methods are not cost-effective to monitor a large enterprise network with millions of concurrent flows, tens of thousands of internal and external hosts, and several tens of Gbps throughput in real-time. As for the detection of network attacks on enterprise assets, existing solutions such as Next-Generation-Firewall (NGFW) and Intrusion Detection System (IDS) often use manual thresholds and static signatures that only focus on certain critical assets directly operated by the organizational IT departments. Operators select suitable detection rules, thresholds, or signatures for a list of enterprise IP addresses each of which may have different attack surfaces. To this end, enforcing proper configurations often requires a comprehensive understanding of connected hosts – it is quite challenging for a large enterprise network having tens of thousands of internal devices with diverse vulnerabilities. In addition, given their limited computational resources, providing fine-grained network telemetry (*e.g.,* at flow-level) of all enterprise assets that require protection is not scalable. Thus, collateral damage on legitimate communications are often introduced when mitigating distributed network attacks.

In this thesis, I present my efforts in advancing the above-mentioned state-of-the-art to address the two above key problems of enterprise network security, *i.e.,* host behavioral monitoring and distributed attack detection. For each problem, I start from domain name system (DNS) protocol that only takes a minor fraction (*i.e.,* less than 0.1%) of traffic volume but plays an utmost important role in network communications, and then extend my scope beyond DNS by developing generic methods applicable to all network traffic across an enterprise.

Legacy solutions often use static configurations and manual thresholds to track host roles and detect network attacks. Such methods struggle to precisely capture the traffic profile of hosts and differentiate attacks from benign instances. In this thesis, I leverage big data analysis and machine learning (ML) techniques to develop statistical models that automatically learn from the empirical network traffic patterns of two representative enterprises, namely, a large-sized university (UNSW Sydney) and a governmental research institute (CSIRO). Through extensive data mining, model tuning, training, and evaluation processes, my developed ML schemes outperform their counterparts in precise and accurate host classification and attack detection. In addition, existing systems trade off the granularity of network telemetry for scalability. Instead of accepting this trade-off, I prototype my systems using network telemetry that is dynamically collected via programmable switches. The scope (*e.g.,* protocol and IP address) and resolution (*e.g.,* packet-, flow-, or host-level) of telemetry are orchestrated in real-time to provide sufficient fine-grained statistics for host classification and attack detection while guaranteeing the operational scalability in a large enterprise network.

1.1 Thesis Contributions

In this thesis, I have made four significant contributions in the field of enterprise network security:

1. First, I develop an automatic method to identify enterprise DNS assets and continuously track their cyber health. Through a comprehensive analysis of over 1 billion DNS packets collected from two representative enterprise networks, I reveal the network, functional, and service properties of DNS packets in both organizations, and highlight normal or anomalous profiles of enterprise hosts. I articulate key DNS behavioral attributes and develop unsupervised machine learning models that identify DNS assets including authoritative name server, recursive resolver, mixed DNS server, and end-hosts that are both public Internet facing or behind NAT. I then develop metrics for tracking DNS traffic health of classified assets that enable operators to detect various DNS anomalies such as improper configuration, DDoS, reflection attacks, and exfiltrations. On a 32-day dataset, my prototype classifies more than 100 DNS assets across the two organizations, some of which are operated by sub-departments thus unknown to the respective organizational IT depart-

ments. Moreover, a non-negligible fraction of them were involved in various types of undesirable DNS behavior. The passive analysis methods help organizations to identify and eventually fix cyber risks associated with their DNS assets (Chapter 3).

2. Second, I develop a systematic approach to classify enterprise hosts by modeling their real-time network behaviors. To understand traffic profiles of enterprise hosts with diverse functionalities and communication patterns, I analyze traffic traces of more than 3 billion packets collected from a large enterprise network. I then propose a multi-grained scheme to classify enterprise hosts into either fine-grained functionality types (*e.g.,* web proxy and mail server) or coarse-grained transport-layer types (*e.g.,* TCP-dominant server). I articulate and evaluate a comprehensive set of host-level behavioral attributes, and train supervised ML models used by the scheme with accuracies of more than 98%. I prototype my approach as a practical system that continuously tracks asset types and reactively collects packet-level telemetry of hosts with potential anomalous behaviors through programmable flow rules for fine-grained post hoc analysis. A one-month campus deployment of my system demonstrates its ability to identify thousands of typical and non-typical assets, track their utilization, and isolate anomalous behaviors related to network attacks. It is proved that my system is scalable to be operated over multiple 10 Gbps links of a large enterprise network and provide visibility for IT departments to effectively track their assets and locate potential anomalies (Chapter 4).

3. Third, I develop a system to detect distributed DNS attacks on enterprise hosts that is effective in isolating external attackers who may be stealthy and distributed. I start by analyzing over 400 million DNS packets from two enterprise networks to highlight profiles of distributed DNS attacks and behaviors of malicious external hosts. I then design a hierarchical graph structure that employs key DNS traffic attributes and well-trained anomaly detection models to

detect anomalous DNS profiles of external entities at various levels of hierarchy with more than 99% accuracy. I prototype my method as a real-time system and demonstrate its merits in detecting distributed DNS attacks by comparing its one-month worth of deployment insights against public blocklists and firewall logs. Results show that my prototype can detect stealthy distributed DNS attacks that are missed by legacy firewalls, capture aggregation profiles of distributed attackers, and have satisfying operational performance in a large enterprise network. (Chapter 5).

4. Fourth, I design and implement a system (named PEDDA) that detects distributed network attacks on enterprise assets effectively at fine-grained flow-level and is also practical to process all enterprise traffic. To formally understand the performance bottlenecks of legacy attack detection systems, I model the traffic processing of legacy static attack detection solutions and formulate its time complexity mathematically. Driven by the insights, I develop a conceptual multi-stage progressive inference architecture that detects distributed network attacks through a series of telemetry stages orchestrated by the dynamic control of programmable networks. To guarantee operational robustness, the granularity of each telemetry stage is adjusted by an optimization process. I then build a proof-of-concept prototype using three inference stages that identify active enterprise hosts, victims, and distributed attackers with malicious flows. The system is evaluated using real network traffic traces from a large university network and ground-truth DDoS attacks. Results show that PEDDA outperforms its counterparts in detecting distributed network attacks effectively at fine-grained flow-level while practical (around 26% CPU and 5% RAM usage on a typical blade server) to be deployed for a large enterprise network (Chapter 6).

1.2 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2, presented in [1], surveys related literature to discuss the taxonomy of distributed attacks on enterprise networks, reviews current state-of-the-art solutions and advances in asset network behavioral monitoring and distributed attack detection, and highlights the potential of emerging paradigms including ML techniques and programmable networks. Chapter 3, presented in [2, 3], elaborates my methods in mapping enterprise DNS assets and tracking their cyber health. In chapter 4, presented in [4], I develop a real-time SDN system to classify enterprise hosts for their fine-grained functionality types or coarse-grained transport-layer behaviors and isolate anomalous changes of host network behavior for packet-level inspection. In chapter 5, presented in [5], I propose a hierarchical architecture that profile DNS volumetric queries from external hosts and various aggregated levels (*e.g.,* subnets) to detect distributed DNS attacks. Chapter 6, presented in [6], describes my design and implementation of PEDDA system leveraging a multi-stage inference architecture to progressively detect distributed network attacks on enterprise assets. Chapter 7 concludes this thesis and discusses potential future directions.

Chapter 2

Literature Review on Enterprise Network Security: Asset Behavioral Monitoring and Distributed Attack Detection

Contents

Enterprise networks that host valuable assets and services are popular and frequent targets of distributed network attacks. In order to cope with the ever-increasing threats, industrial and research communities develop systems and methods to monitor the behaviors of their assets and protect them from critical attacks. In this chapter, I systematically survey related research articles and industrial systems to highlight the current status of this arms race in enterprise network security. First, I discuss the taxonomy of distributed network attacks on enterprise assets including distributed denial-of-service (DDoS) and reconnaissance attacks. Second, I review existing methods in monitoring and classifying network behavior of enterprise hosts to verify their benign activities and isolate potential anomalies. Third, the state-of-the-art detections against distributed network attacks from external attackers are elaborated, with highlights on their merits and bottlenecks. Last, as programmable networks and machine learning (ML) techniques are becoming widely adopted by the community, their current applications in network security are discussed to inspire future research especially for enterprise networks.

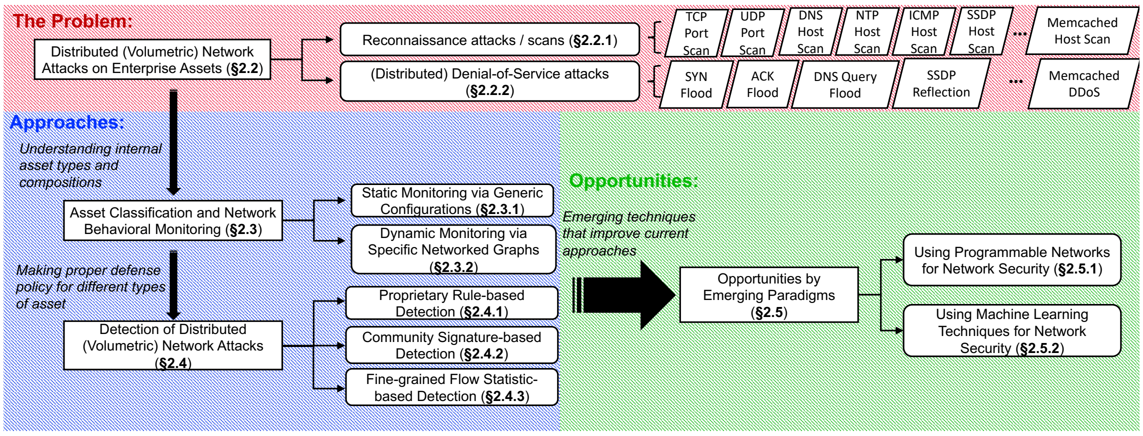

The Problem:
Distributed (Volumetric) Network Attacks on Enterprise Assets (§2.2)
Reconnaissance attacks / scans (§2.2.1)
(Distributed) Denial-of-Service attacks (§2.2.2)
TCP Port Scan
UDP Port Scan
DNS Host Scan
NTP Host Scan
ICMP Host Scan
SSDP Host Scan
Memcached Host Scan
SYN Flood
ACK Flood
DNS Query Flood
SSDP Reflection
Memcached DDoS
Approaches:
Understanding internal asset types and compositions
Asset Classification and Network Behavioral Monitoring (§2.3)
Static Monitoring via Generic Configurations (§2.3.1)
Dynamic Monitoring via Specific Networked Graphs (§2.3.2)
Making proper defense policy for different types of asset
Detection of Distributed (Volumetric) Network Attacks (§2.4)
Proprietary Rule-based Detection (§2.4.1)
Community Signature-based Detection (§2.4.2)
Fine-grained Flow Statistic-based Detection (§2.4.3)
Opportunities:
Emerging techniques that improve current approaches
Opportunities by Emerging Paradigms (§2.5)
Using Programmable Networks for Network Security (§2.5.1)
Using Machine Learning Techniques for Network Security (§2.5.2)

Figure 2.1: Key topics covered in this survey.

2.1 Introduction

Enterprises such as universities and research institutes host critical data and offer public accessible services through their networks. Thus, they often become popular targets of distributed network attacks that actively probe vulnerabilities of assets and paralyze their services. With practical defense appliances (*e.g.,* firewalls and intrusion detection systems) installed and operated by enterprise IT departments, network attacks are becoming well distributed in sources and agile in attacking patterns to bypass such static detection and increase their effectiveness. To be more specific, a sophisticated network attack usually employs hundreds and thousands of botnet devices that are diversified for their geolocations and types (*e.g.,* Internet-of-Things, laptop, and compromised server), each may send malicious traffic with changing patterns and protocols. As an example, during Rio 2016 Olympics, the critical servers from organizations affiliated with the Olympics (such as Brazilian banks and telcos [7]) were targeted by sustained distributed network attacks with mixed traffic types such as TCP-SYN, UDP reflection, DNS, chargen, NTP, and SSDP from millions of compromised devices (*e.g.,* IoTs) all over the world [8]. Successes of distributed network attacks cause severe consequences like service failures, disruption of operations, and degradation of reputation.

Distributed attacks on enterprise networks often consist of two phases, namely reconnaissance attacks (also known as scans) to discover the vulnerabilities of networked assets and distributed denial-of-service (DDoS) attacks that paralyze the targeted victims that are discovered by malicious actors. To cope with the threats, enterprise IT departments are expected to track the devices within their networks to ensure their expected behaviors and enforce attack defense mechanisms that can effectively detect and mitigate attacks on their networked assets without impacting legitimate communications.

There are many mature products for monitoring the network behaviors of enterprise assets and providing protections against distributed attacks via static configurations, such as next-generation-firewall (NGFW) appliances and intrusion detection systems (IDS). These static solutions are practical to be operated in a high-throughput enterprise network but are not effective in providing precise results (*e.g.,* differentiating distributed attackers and malicious flows from their benign counterparts), therefore, it is not surprising that the consequential attack mitigation measures (*e.g.,* randomly drop packets to the victim) introduce non-negligible collateral damage on benign traffic [9]. For instance, typical next-generation-firewalls (NGFW) require users to configure rules that specify the list of focused enterprise assets and the corresponding defense strategies. Such methods are effective in protecting certain key assets by tracking their network activities of several traffic types but fail to capture unknown and complex threats from hosts operated by sub-departments, staff, and visitors. Moreover, the static nature of such methods limits their capabilities in detecting emerging attacks with dynamic and stealthy traffic patterns [10].

Legacy static solutions introduce blind spots that are likely to be exploited by malicious actors and agile attackers. To address this problem, research communities have developed dynamic telemetry methods for network monitoring via flow-level statistics and networked graph structures. Those methods have the ability in providing fine-grained statistics to track each network flow between enterprise assets and external hosts without leaving any blind spot. However, maintaining fine-grained flow-level telemetry unavoidably introduces high computational overheads. Therefore, they are not scalable to be operated for a large enterprise network with hundreds and thousands of hosts that send millions of concurrent flows.

In the meantime, current developments of two emerging paradigms, namely Programmable Networks and Machine Learning (ML) have given new promises in improving the flexibility of network monitoring and accuracy of attack detection. The concept of programmable networks includes network function virtualization (NFV) and software-defined networking (SDN). It changes the static nature of network traffic processing that is often carried by proprietary hardware, instead, dynamic network functions on generic servers and programmable switches are used to achieve high responsiveness and real-time orchestrations. Researchers have leveraged this technology to address the inflexibility problem of legacy network monitoring and protection in various aspects such as real-time defense orchestration for ISP network [11] and elastic control of virtual firewalls [12] that provide inspirations in solving the current problems of enterprise network security. The recent advances of ML techniques that build learning-based models to make accurate predictions on statistical attributes have proven their supremacy in many disciplines such as image processing and language recognition. Despite the existing challenges in applying ML methods for network security [13], researchers have used ML algorithms to develop prototypes that make accurate security inferences on various types of network telemetry (*e.g.,* system logs or packet headers) in different scenarios (*e.g.,* IoT attack or SSH brute-forcing). I believe that their efforts provide valuable lessons for us to address issues associated with asset classification and attack detection in an accurate and precise manner.

In this survey, by systematically reviewing related research articles as well as industry practices, I provide comprehensive guidance towards the current development, challenges, and future directions of asset management and distributed attack detection in the field of enterprise network security. Unlike prior surveys that broadly studied certain attack types and defense mechanisms, I focus on a narrow aspect of distributed volumetric network attacks and their countermeasures that are applicable for enterprise networks. In addition, I also review the potential and challenges to improve the state-of-the-arts by two emerging paradigms (programmable networks and ML). To this end, I summarize the main topics covered by this survey as follows, which is also visually shown in Fig. 2.1. **First**, in §2.2, I reveal the diversity and variety of distributed network attacks including reconnaissance scans and distributed denial-of-service (DDoS) attacks; **second**, in §2.3, I discuss the current development of enterprise networked asset classification and behavioral monitoring via either static or dynamic methods; **third**, in §2.4, enterprise distributed attack detection systems using proprietary rules, community signatures, and fine-grained flow statistics are elaborated; **fourth**, in §2.5, opportunities introduced by the two emerging paradigms, *i.e.,* flexibility by programmable networks and accuracy by machine learning are discussed as to inspire future research.

The related surveys on other different aspects of network security are discussed in §2.6. I highlight several research gaps as valuable future directions in §2.7, and conclude this survey in §2.8.

2.2 Surveying Distributed (Volumetric) Network Attacks on Enterprise Assets

Network attacks that probe, congest, or paralyze enterprise assets such as public-facing servers are becoming distributed in sources, versatile in traffic patterns, and diversity in underlying mechanisms [14–16]. Such attacks often occur sequentially – an enterprise asset is first probed for its availability and vulnerability by recon-

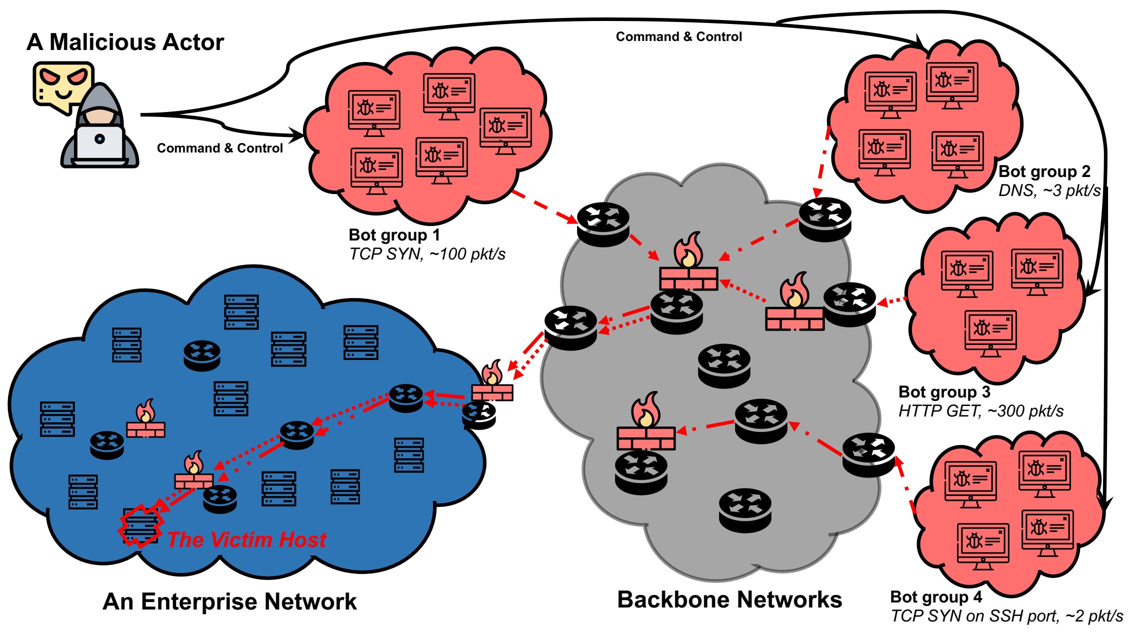


Figure 2.2: An visual example of distributed network attacks on a victim inside an enterprise.

naissance attacks (*i.e.*, host or port/service scans) followed by distributed denial-of-service (DDoS) attacks.

Both scans and DDoS are often operated in a distributed manner instead of from a single source to increase their attack power and make them hard to be blocked. Such distribution is achieved by recruiting botnets that consist of massive compromised devices like personal computers, powerful workstations, public-facing servers, or compromised IoT appliances [17–21]. To avoid detection, malicious actors often split an attack into small segments each performed by a single bot device. For example, in a powerful but stealthy DDoS attack – each bot device only generates low rate traffic which makes it difficult to be differentiated from benign instances. Thus, such attacks are hard to be precisely blocked by identifying all attack sources [22]. I show a visual example of distributed network attacks in Fig. 2.2, where a malicious actor commands and controls four distributed bot groups to attack a victim within an enterprise network. Each group uses different traffic types and rates so that some fraction of the attack traffic (sent by bot groups 1 and 2 in Fig. 2.2) may successfully bypass defense appliances (*e.g.,* firewalls operated in backbone and enterprise networks that are not optimally tuned to detect those stealthy malicious traffic) on the path and reach the victim.

2.2.1 Reconnaissance Attacks

Reconnaissance attacks (also known as scans) are used by malicious actors to construct their understanding of the targeted hosts and services (ports). Those attacks probe the availability of enterprise networked hosts and discover their potential vulnerabilities [23]. The discovered hosts may not only become victims but may also be exploited as attack amplifiers/reflectors to paralyze other victims. For example, a discovered NTP server with high reflection capability (*i.e.,* generate response packets with sizes larger than the received requests) can be used to amplify attack volume in reflection-based DDoS attacks [24].

Apart from the malicious purposes, security researchers also develop tools to identify potential cyber threats faced by enterprises such as open ports and vulnerable services that could be exploited in network attacks. For example, *Nmap* [25] is developed as a comprehensive scanning tool to discover active hosts and ports (*i.e.,* services). To increase the speed and effectiveness of scans, the authors of *Zmap* [26] optimized the scanning process by tuning the probing rate, pre-connection state, and re-transmission, which can probe the entire IPv4 space within 45 minutes. Scanning techniques have evolved to become scalable at 10 Gbps throughput [27] and can perform vulnerability scans towards protocol banners through user queries [28]. Reconnaissance attacks have also been studied for certain scan types such as critical cyber-physical infrastructures [29] and DNS utilities [30].

To detect reconnaissance attacks targeting hosts and services, the community has developed various methods such as tracking port scanners on the IP backbone [31], detecting subtle port scanning via interactive visualization [32], disrupting reconnaissance attacks via address mutation [33], constructing distributed network telescope to capture scanners [34], and optimizing backscatter [35] technique for scan detection in massive IPv6 address space [36]. However, according to [37, 38], few enterprise has adopted proper defensive measures in practice, thus, service and host scans are still prevalent in today's Internet. It leads to the exposure of service and device vulnerabilities (*e.g.,* Linksys routers, OpenSSL, and NTP). As a consequence, the exposed hosts may be utilized by malicious actors on the Internet to generate/reflect attacks or become direct victims in future attacks.

2.2.2 Distributed Denial-of-Service Attacks

Distributed denial-of-service (DDoS) is one of the most popular network attacks on the Internet [39]. Attackers command a massive number of controlled devices to generate an excessive amount of traffic on victims to exhaust their network or computational resources. Both large-sized global enterprises and small-sized local industries are frequent targets for such attacks. As already shown in Fig. 2.2, malicious actors may choose to flood the targeted victim directly from botnet devices using various protocols and headers (*e.g.,* HTTP, ICMP, and TCP-SYN). Besides, they may also choose to launch a reflection-based DDoS that have larger attack volumes – bot devices send packets with crafted IP addresses (as that of the victim) to the discovered reflectors (*e.g.,* DNS and NTP servers), these reflectors will then respond to the victim with larger packet sizes.

DDoS attacks are becoming more complex, distributed, and agile. Their characteristics are extensively studied in the prior research literature. First, according to [40], DDoS attacks are becoming complex in protocols and traffic types, and the involved botnets are likely to be independent. Such trends introduce difficulties in isolating malicious traffic and attack sources. Second, the increasing adoption of cyber-physical devices (*e.g.,* IoTs) brings new vulnerabilities and attack surfaces yet to be addressed [41]. Therefore, a significant number of IoT devices connected to the Internet have been compromised as botnet and caused more powerful and frequent DDoS attacks on a global scale [42]. For example, in late 2016, Mirai [43], an IoT malware that hijacked hundreds of thousands of IoT devices has caused many unprecedented DDoS attacks globally. During an attack, each compromised IoT device generates malicious traffic at a low rate, making them hard to be differentiated from benign traffic. Third, DDoS attacks are becoming more dynamic and agile in traffic patterns to evade detection. As identified in [44], they are usually launched with changing temporal and spatial patterns to bypass detection, which is quite effective for static rule-based and signature-based detection methods. Botnets of different families also work collaboratively, and the same bot might adapt its attacking strategy provided by different malware families [45]. Last, the concept of DDoS-as-a-service is becoming popular as it lowers the barrier for generating an effective distributed attack [46]. Botnet owners can lease their controlled devices for financial benefits so that malicious actors with fewer resources (*e.g.,* controlled bot) can rent their large botnet to launch powerful attacks.

2.2.3 Highlights

I now summarize three highlights in this section.

First, network attacks such as DDoS and scans are becoming distributed by recruiting botnets to generate attack traffic from different logical affiliations (*e.g.,* subnets) and geolocations, complex by leveraging a wide set of protocols and vulnerabilities, and dynamic by shifting active bot groups or traffic patterns randomly to bypass possible detection. All the above characteristics increase the difficulties in effectively detecting distributed attacks.

Second, potential vulnerabilities of network-connected hosts (*e.g.,* BYOT devices, enterprise servers, or IoTs) within an enterprise network may be identified and exploited by malicious scripts (*e.g.,* links contained in phishing emails) or malware. Such compromised devices are used as bots to perform further infections within its network or participate in attacks on other networked assets. Thus, monitoring network traffic behaviors and enforcing proper security management are important tasks for IT departments. Therefore, later in §2.3, I will discuss related tools and techniques for asset network behavioral monitoring.

Third, apart from malware infections and misuses, assets such as servers and databases within an enterprise network may be direct targets of distributed attacks. If such an attack happens, public-facing servers may not respond to requests from external clients since their networking and computational resources are exhausted. In addition, network vulnerabilities of hosts in an enterprise network may be exposed and recorded by external hackers for further cyber-crimes. Therefore, defending against distributed attacks on enterprise assets is a critical job for security operations. In §2.4, I will elaborate on state-of-the-arts enterprise attack detection systems and mechanisms.

2.3 Surveying Asset Classification and Network Behavioral Monitoring

To cope with the increasing trends of distributed network attacks targeting or utilizing enterprise assets, IT departments need to classify and monitor the roles and network behavioral patterns of hosts connected to their networks (*i.e.,* fingerprinting enterprise host types and their behavioral patterns) so that proper security enforcement could be installed.

However, this task is not straightforward as enterprise hosts have diverse and complex functionalities and behaviors. For example, an enterprise has servers of various types that serve internal or external clients; visitors and staff may have

Table 2.1: Classifying host types in a large enterprise network by DNS names.

Asset type	**# hosts**
Website server	61
Authoritative name server	15
VPN gateway server	13
Remote computing platform	16
File storage server	14
Mail exchange server	18
DNS recursive resolver	7
Web proxy	4
NAT gateway	256
Personal computer and BYOTs	1961
Other unclassified or minor host	18920

their personal devices (*e.g.,* mobile phones and laptops) connected through wireless gateways; and IoTs such as smart cameras and sensors may also be installed in a typical enterprise network [47]. Let us take a glance at Table 2.1 that shows the total number of ten popular host types identified by their DNS names in a large university network. As shown by the last row in Table 2.1, there are also many other unclassified and minor host types such as LDAP server and Redis proxy which are hard to enumerate. I note that those identifiable assets are only accountable for less than 10% of active hosts in the enterprise, and the functionality of the remaining hosts is mostly unknown to the university IT department.

The massive amount of connected devices unavoidably introduce security problems and threats to an enterprise network. Devices that are carried by visitors and staff may be infected by malware and then conduct malicious network activities [48], which may not be fully understood by organizations [49, 50]. Besides, insecure network management, improper configurations, and unprofessional operations of IT infrastructures (*e.g.,* public-facing servers) may give external attackers opportunities to hijack or penetrate internal hosts for malicious purposes. As for organizations with less-restricted network management like universities and research institutions that allow sub-departments to configure their own IT infrastructures, this problem is more prevalent since their IT departments may not be fully aware of the roles of all connected network assets.

To classify roles and monitoring network behavioral profiles of enterprise hosts, there are many solutions developed by both industrial and academic communities, which can be categorized into either static monitoring via generic configurations or dynamic monitoring via specific networked graphs.

2.3.1 Static Monitoring via Generic Configurations

Current practical solutions for enterprise network asset classification and monitoring are usually achieved by static configurations such as access control list (ACL), security rules, and system records in their management devices like border firewalls, DHCP gateways, DNS servers, and other specialized commercial management platforms. Using such methods, IT departments can set up certain specifications or limitations on the monitored hosts to regulate their network traffic and detect potential anomalies. An IT department that is fully aware of all assets connected to its network may set up strict configurations to stop illegitimate communications [51, 52]. Network traffic not following those configurations will be marked as abnormal, thus, trigger further defense actions such as alerts and mitigation. For example, all non-HTTPS traffic towards an enterprise HTTPS server could be blocked by the broader firewall as they are unwanted by the server, or inbound DNS packets are only allowed if their destinations are enterprise DNS servers. To protect a critical asset operated within an enterprise, the IT operator may also set up a rate limit for the host on its management platform. Whenever the host receives network traffic with rates higher than the limit, the management system (*i.e.,* border firewall) will partially or fully drop those inbound packets to prevent potential volumetric damage on the protected host.

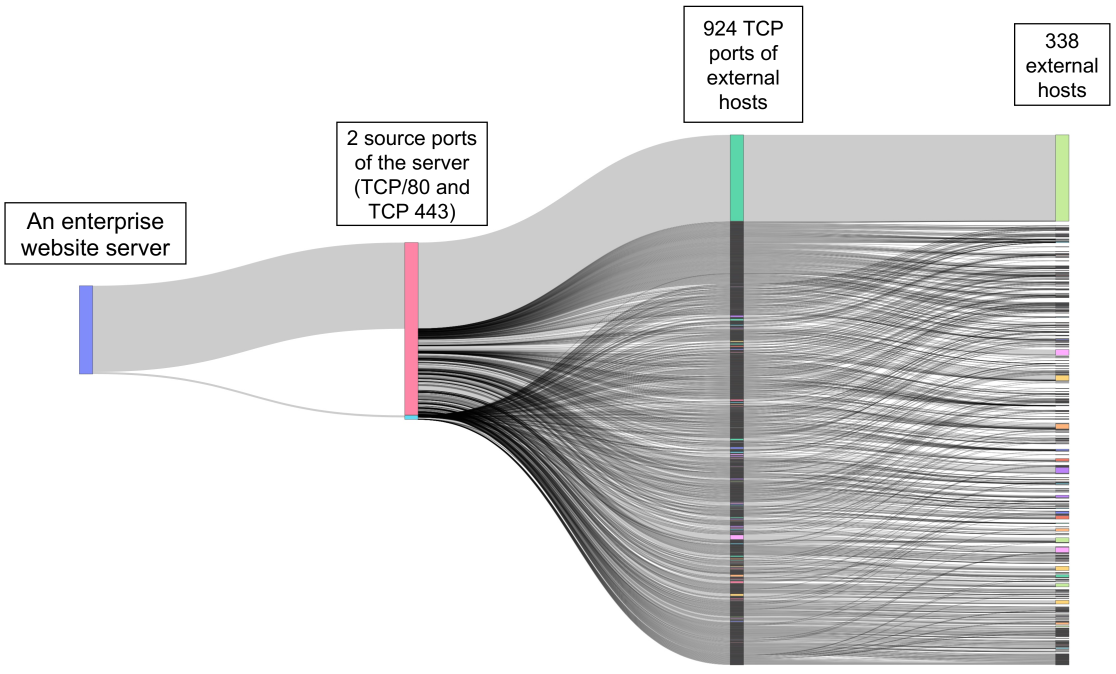
Static configurations enable administrators to manage and monitor their enterprise assets by specifying their generic network profiles. However, with the explosive growth of network applications that are developed using different protocols and the adoption of networked devices with heterogeneous architectures, designing comprehensive and proper generic configurations becomes increasingly difficult for IT departments, especially for those loose-federated enterprise networks [53]. As highlighted in [54–57], configuring specifications or rules for a large enterprise network with complex host composition is error-prone, and potential misconfigurations may introduce expensive operational costs in bug fixing and conflict resolving. This problem is even worse with the adoption of BYOTs and IoTs which are massive in amount and diverse in communication patterns [58] – the complex behavioral profiles of such hosts are hard to be precisely concluded by several generic configurations. Therefore, asset monitoring via generic configurations tailored for each device type inevitably leaves many blind spots and becomes impractical during operations [59, 60].

Besides, the roles and behaviors of connected hosts in nowadays' enterprise networks are quite dynamic. For example, sub-departments may operate multiple public-facing services (*e.g.,* DNS and website) with different network profiles on the same physical machine; existing services may get terminated and new functionalities may be added in an on-demand fashion. As a result, static asset monitoring methods cannot responsively and automatically cope with the changes of connected assets and their network behaviors [61–63].

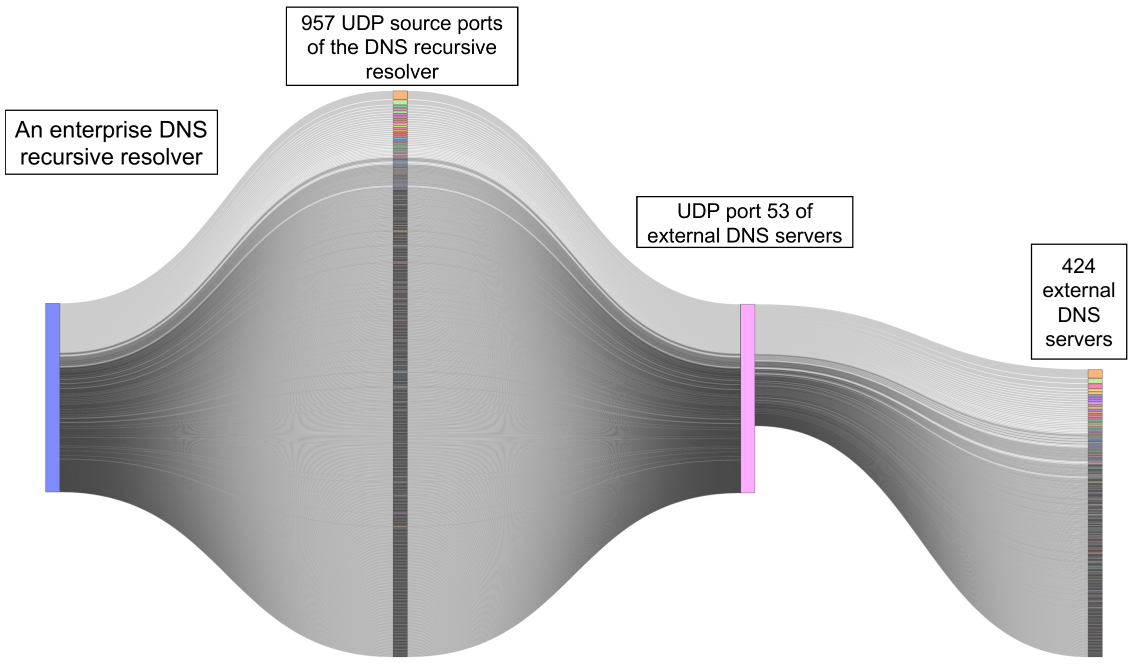
As motivated by the current problems in asset classification and monitoring, researchers have developed dynamic methods using specific networked graphs, which will be discussed next.

2.3.2 Dynamic Monitoring via Specific Networked Graphs

To monitor activities of enterprise hosts at fine-grained visibility, networked graphs have been used in prior works to profile host types each may display unique communication patterns. For example, in Fig. 2.3, I show the flow graphs (in the format of



(a) A website server.



(b) A DNS recursive resolver.

Figure 2.3: Sankey diagrams: example traffic profiles of two enterprise assets using 1000 continuous flows.

Sankey diagram) of two enterprise assets in my university network (*i.e.,* a website server and a DNS recursive resolver) using their 1000 continuous flows. The website servers use only two TCP ports (*i.e.,* `TCP/443` and `TCP/80`) to communicate with a large range of TCP ports on external hosts, while the DNS recursive resolver sent traffic via a large scope of its UDP ports to only `UDP/53` on external DNS servers. I note that about 20% flows of the website server are from one port of an external host. After looking into their flow contents, I confirm that those TCP flows only contain TCP handshakes and are likely to be TCP-SYN attacks.

Using Graphs for Host Classification

Based on such facts, in [64], the authors pointed out that the type of enterprise host can be inferred from its IP- and port-level activities. They used networked graphs that profile communications between hosts and ports to classify host roles such as HTTP servers, DDoS attackers, and P2P clients. To effectively monitor communication patterns of hosts, the authors of [65] proposed visualization techniques using an interpretable graph that help operators to easily classify networked entities. Apart from the legacy graphs that only cover flow profiles, researchers have developed advanced graph structures that can model network communications with more descriptive features, such as attributed graph models in [66]. The graph structure is generated to cover both network structural properties and correlated attributes which hold both efficiency and accuracy when classifying host roles in real-world networks.

Using Graphs to Model Host Dependency

Understanding host dependencies and classifying roles of networked entities at scale from complex graphs require extensive tuning and optimization of existing statistical methods which are computationally expensive. In [67], the authors analyzed dependency graphs in terms of their computational resource consumption and expressiveness in modeling network attacks. Authors of [68] used the method of connection graph analysis to discover cooperating hosts in P2P networks, which start from a single known node to identify other associated hosts. Their methods processed unified NetFlow data and were demonstrated to have short processing times, which could scale for large networks. Besides, statistical methods such as clustering algorithms are a natural fit for grouping host types and identifying different types of behaviors in a networked graph. In [69], the authors used bipartite graphs with one-mode projections that model network traffic between hosts, and developed clustering algorithms to identify behavior groups and typical application types. As in [70], the authors grouped IP addresses within the same enterprise network with strong inter-IP connectivity (*i.e.,* similar behavior) as the basis of network monitoring and management, which significantly reduced the number of tracked entities for scalability.

Using Graphs for Anomaly Detection

It is noted that network asset monitoring with fine-grained graphs is especially useful for security purposes such as locating infected hosts and identifying involved parties in distributed attacks [71–74]. For example, authors in [50] mentioned that securing a large enterprise network that has a fluidic structure is complicated and error-prone technically. Therefore, they proposed a probabilistic graph to model the defense strategies of a network to reduce the possibility of successful attacks that target improper and vulnerable network configurations. *SpotLight* [75] achieved accurate detection of anomalies in streaming graphs that describe network communications between entities. The anomalies (*e.g.,* port scans and DoS) are identified by the sudden changes in large dense subgraphs. It leveraged randomized sketching algorithms to make a cost-effective inference with optimal memory consumption. *Noracle* [76] can classify service roles of networked entities and identify anomalous behavioral changes by applying stochastic block models on networked graphs. Whereas *TRACE* [77] builds a distributed enterprise-wide causal graph to track network activities between enterprise hosts for advanced persistent threat (APT) detection.

2.3.3 Highlights

In summary, asset monitoring using static configurations on border firewalls and management middleboxes is the de facto method in industrial practices. Such methods are practical in terms of computational costs as they usually do not maintain complicated real-time data structures for a large number of monitored entities. This method prioritizes practical deployment but makes it impossible for fine-grained classification (*e.g.,* flow-level) and effective monitoring of host behaviors that are dynamic or unknown to IT departments.

In contrast, dynamic network behavioral monitoring by specific graphs is proven to be effective in providing comprehensive visibility and granularity of network traffic to classify connected assets and detect potential anomalies. However, the use of complex graphs incurs high computational costs that make such methods not scalable to be deployed in a large enterprise network with a massive number of active hosts and concurrent flows.

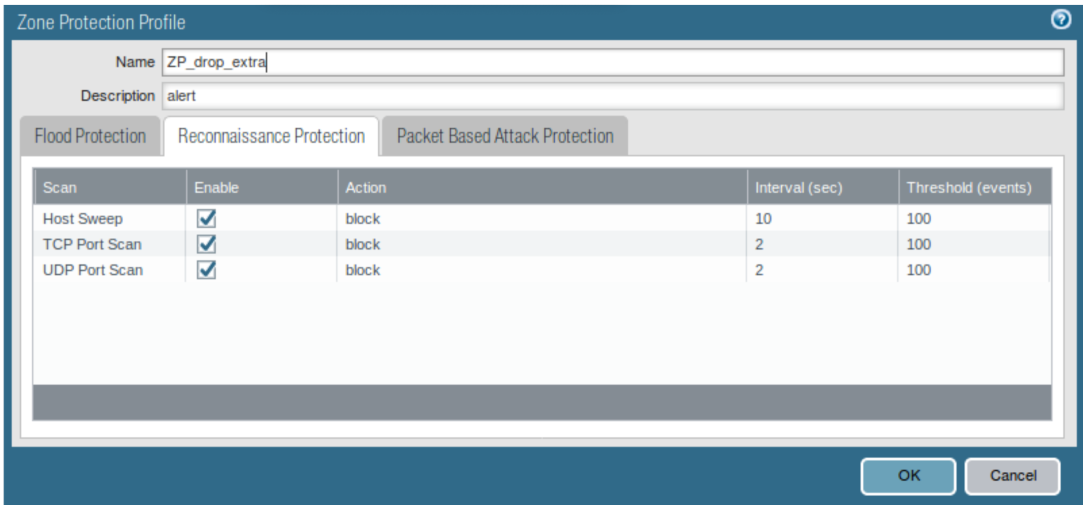
2.4 Surveying Enterprise Distributed Network Attack Detection

Detecting distributed attacks is critical for enterprise network operations. The security community comes up with solutions from various deployment considerations. For those attacks that aim to congest the entire edge link of an enterprise network by Giga or Tera bit-per-second traffic throughput, handling attack at ISP-level (near-source and on-the-path) appear to be the only choice [11, 78–81]. As for distributed attacks targeting certain enterprise assets, that is, the focus in this survey, detection systems operated by the targeted enterprise (near-victim) is proven to be effective [78]. Therefore, enterprise IT departments usually set up in-line security middleboxes near their network edge that connect internal networks and the public Internet. To this end, attack detection policies can be implemented for each critical enterprise server [82] that is vulnerable to network attacks. Such enterprise-level attack detection solutions can be categorized into three types in terms of their methodologies, including threshold rules, traffic signatures, and flow statistics, which are comprehensively reviewed as follows.

2.4.1 Proprietary Rule-based Detection

Simple Thresholds

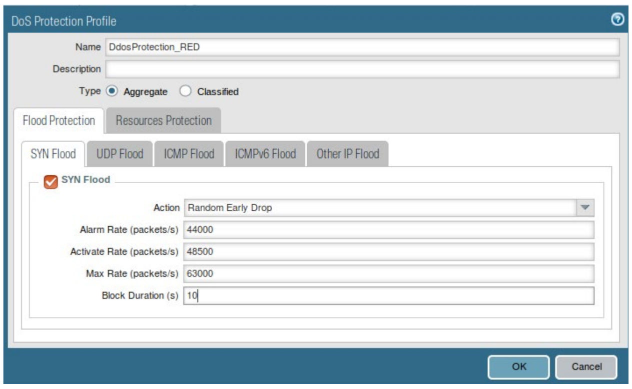
Rule-based (also known as query-, specification-, or configuration-based) distributed attack detection is the most popular type in the market. Proprietary hardware products such as next-generation-firewall (NGFW) deployed at an enterprise's border uses such mechanisms as the basis of attack and intrusion detection. Users (*e.g.,* IT departments) set up a list of rules that define access policies of certain connected hosts, which indicate the normal and abnormal traffic levels by thresholds. In Fig. 2.4, I present three screenshots of configuration pages on a firewall appliance (by Palo-Alto Networks) operated in a large-sized university. They are for defenses against reconnaissance attacks (Fig. 2.4(a)), DDoS attacks via SYN flood (Fig. 2.4(b)), and DDoS attacks via UDP flood (Fig. 2.4(c)), respectively. For the reconnaissance protection, as shown in Fig. 2.4(a), IT departments who want to protect their assets from host reconnaissance or port scans may set up a security rule to block all external IP addresses that have sent packets to more than a certain number of hosts within the monitored network. As for the protection against DDoS via SYN flood in Fig. 2.4(b), an administrator may set up thresholds for inbound TCP-SYN



Name: ZP_drop_extra
Description: alert

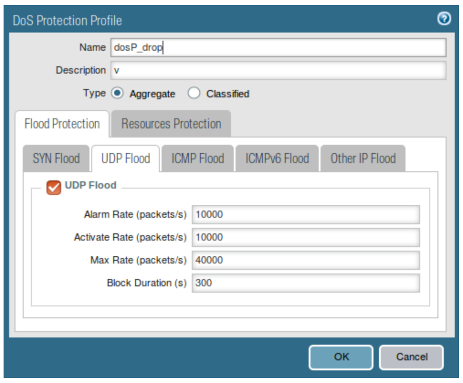
Scan	Enable	Action	Interval (sec)	Threshold (events)
Host Sweep	✓	block	10	100
TCP Port Scan	✓	block	2	100
UDP Port Scan	✓	block	2	100

(a) Reconnaissance attack.



Name: DdosProtection_RED
Description:
Type: [✓ Aggregate] Classified
✓ SYN Flood
Action: Random Early Drop
Alarm Rate (packets/s): 44000
Activate Rate (packets/s): 48500
Max Rate (packets/s): 63000
Block Duration (s): 10

(b) DDoS – SYN flood.



Name: dosP_drop
Description: v
Type: [✓ Aggregate] Classified
✓ UDP Flood
Alarm Rate (packets/s): 10000
Activate Rate (packets/s): 10000
Max Rate (packets/s): 40000
Block Duration (s): 300

(c) DDoS – UDP flood.

Figure 2.4: Configuration pages of a Palo-Alto firewall appliance for distributed attack detection.

packet rates towards certain IP zones, surpassing this threshold indicates volumetric anomalies and corresponding alerts or actions would be triggered. As shown in Fig. 2.4(c), similar detection and mitigation thresholds can also be configured for UDP-based DDoS attack.

Although legacy proprietary security middleboxes leveraging administrator-defined rules have been widely deployed by the industry for distributed attack detection, their mechanisms are quite simple via several thresholds thus not flexible to cope with emerging security needs such as detecting distributed attack sources with versatile traffic patterns. Besides, rules and policies for security appliances manufactured by different companies are not compatible with each other, which introduces practical difficulties for operators to express complicated logic effectively [83].

Expressive Queries

Considering the above problems, researchers have developed expressive rule-based security middleboxes that support a larger scope of detection logic than simple thresholds. To make network attack detection sustainable and versatile under fast-changing environments, the authors of *Marple* [84] designed a query language to perform monitoring tasks via key-value store primitives on programmable P4 switches. To make a rule-based security mechanism suitable in coping with cyber-attacks that involve various logical steps and a large number of networked entities, *SAQL* [85] was developed as a stream-based query system that provides an anomaly query engine for users to specify their complicated detection logic using domain-specific languages. By leveraging both programmable P4 switches and software stream processors, *Sonata* [86] was proposed as a network telemetry system that is scalable and expressive in performing security tasks (*e.g.,* detection of SSH brute force, port scan, DDoS, or slowloris attack) with fewer configurations compared with existing systems. Although those research ideas still have a far way to go before being fully accepted by the industry, they are valuable steps towards a low cost, easy to upgrade, and expressive rule-based detection system.

While rule-based security systems are quite popular in the industry, configuring effective and error-free specifications require administrators to have sufficient security expertise and a full vision of connected assets within their networks, without which they can hardly set up proper thresholds, queries, or actions. Besides, for medium or large enterprises that have complex host compositions and behaviors, managing manual configurations are challenging in handling unavoidable redundancies, logical conflicts, and errors.

Performance Evaluation and Optimization

To optimize the placement of security rules/queries and solve potential redundancies, an extensive amount of research efforts have been made by the community. The authors of [87] conducted experiments to evaluate performance degradation of placing firewall policies such as to permit or deny certain services unless explicitly specified. They concluded that the setup of firewall rules (*e.g.,* sequence of rules and their applied subnet range) has a significant impact on networking metrics such as latency and throughput, thus, optimization of firewall technologies is critical in reducing performance losses. In [88], a quantitative analysis of firewall configuration data draws the conclusion that corporate firewalls are often not properly configured to provide sufficient security protection. Although examples and guidelines are presented by vendors, network administrators still face great challenges in following them manually. To better understand the performance issues of rule-based firewalls, the authors of [89] constructed a queuing model with a Markov chain to investigate key performance features of firewalls when handling normal or DoS traffic flows. In [90, 91], performance bottlenecks such as CPU and memory usage under different network conditions such as traffic rate, packet size, and the number of connections are extensively studied.

Considering the above operational difficulties and performance bottlenecks of rule-based solutions, optimization methods have been developed using various techniques. As legacy firewalls check the received packets against each placed rule, the increasing number of firewall rules unavoidably delays the overall processing time, in [92], the authors proposed a data mining approach to predict hit probabilities of rules which significantly reduces packet processing time up to 40%. To tackle redundancies in rule placements, the authors of [93] proposed an optimization algorithm to locate and reduce redundant configurations within an enterprise firewall. The work described in [94] designed a stateful firewall architecture that is able to classify network traffic according to their application types, each is mapped to a customized processing pipeline to achieve better performance on CPU utilization, throughput, and queue delay. In [95], a hash-based packet classification algorithm was developed to significantly reduce the delays caused by the rule matching processing in a typical firewall appliance. Although there are a handful of prior research for optimizations on errors and performance degradation introduced by redundant firewall rules via manual configurations, according to [96], rule-based firewall performance issues are still key concerns yet to be solved.

2.4.2 Community Signature-based Detection

With the increasing complexity of attack types and vectors, implementing effective security rules by administrators has become quite challenging than ever before. To ease this pain point, the security community developed various software intrusion detection systems (*e.g.,* Bro [97] and Snort [98]) that do not require complicated configurations. Instead, users could simply import security signature files that contain features of malicious traffic articulated by experts and researchers in the open-source security community.

Merits: Compared with rule-based detection via proprietary systems, signature-based attack detection usually leverages CPU-based generic software that supports highly flexible traffic processing functions. In addition, software IDS does not require a tedious process for upgrading security functions including negotiations between users and vendors, instead, administrators can customize functionalities and deployment settings. As stated in [10], hardware appliances are designed for high performance (*e.g.,* sustain Tbps traffic) thus sacrifice operational flexibility in dynamic network environments, while software-based systems tackle those limitations by elastically scaling or replacing detection functions based on operational needs and traffic composition.

Problems: However, there are also problems in using community signature-based IDS software for a practical operation. First, creating security signatures that capture traffic patterns and features of diverse attacks is a complicated process that introduces high labor costs. Besides, the threat surfaces of each network may be quite different, signatures developed by third-parties may not be reliable and directly applicable to an enterprise network, thus, have low detection effectiveness. In addition, such systems are often operated as software tools that incur high computational costs (thus not scalable) when processing high throughput enterprise traffic. Around the three problems, researchers have developed methods in automatic generation of signatures, benchmarking and increasing detection effectiveness, and improving the scalability of signature-based IDS.

Automatic Generation of Signatures

Many research works aim to develop automatic methods that generate reliable signatures. C. Kreibich *et al.* [99] presented a system to automatically generate signatures for pattern matching and protocol conformance check. To this end, they set up honeypots to capture malicious network traffic signatures passively. Security signatures may be evaded by smart attackers who vary payload contents of malicious packets, to address this issue, *Polygraph* [100] is proposed to automatically generate signatures that contain multiple disjoint content sub-strings for polymorphic worms (*i.e.,* an example attack that vary its packet payloads frequently). *AutoRE* [101] captures spam emails and their involved entities (*e.g.,* bot members) by extracting signatures from spam traffic that does not require a labeled dataset and extensive labor efforts for training purposes.

Detection Effectiveness

To improve the detection effectiveness of signature-based detection systems, researchers have identified various related problems and proposed corresponding solutions. In [102], S. Patton *et al.* reported the "Squealing" vulnerability of signature-based IDS. Attackers may craft malicious packets according to the observed signatures on a targeted IDS to generate high false-positive events which makes the alerting system near useless. R. Sommer *et al.* [103] observed that the legacy signatures using byte sequences suffered from high false-positive rate due to the dynamicity of attacks. To address this issue, they designed a signature engine based on Bro IDS [97] that can generate signatures with enhanced visibility such as dependencies of networking events (*e.g.,* requests and replies). The works described in [104] and [105] compared the accuracy and performance of IDS designed for single-threading and multi-threading computing environments (*i.e.,* Snort and Suricata respectively). They concluded that the latter IDS have higher accuracy under a multi-core setup, while within a single-core networking system, the former IDS achieved fewer false negative instances. Besides, according to [106, 107], the adoption of emerging network ecosystems such as IoT and sensor networks also makes legacy security signatures less effective in capturing malicious activities, as they exhibit different traffic patterns compared with typical networked hosts.

Scalability Issues

Compared with well-designed hardware appliances, software-based IDS generally incur high computational costs thus not scalable under high-throughput environments [10], ineffective design of software components makes this problem even worse [108]. Therefore, signature-based IDS on software platforms are mostly used by enterprises with small network sizes and low traffic throughput. Researchers use various techniques to increase the scalability of software IDS. First, the concept of distributed computing has been exploited by many prior works. *The NIDS cluster* described in [109] used distributed computational nodes with optimized coordination approaches to achieve software-based stateful intrusion detection with high performance. The authors of [110] proposed a domain-specific model that distributes traffic analysis across different processing units with specific functions to achieve both scalability and detection efficiency on multi-core hardware. Researchers have also developed methods to reduce the overheads by signature-matching. For example, the work in [111] developed an alphabet compression table that can combine distinct input signature symbols that have identical behavior into a single one to reduce memory usage. *O3FA* [112] was proposed to achieve packet ordering and flow reassembly during pattern-matching phases with low buffer consumption, which is particularly useful in reducing computational overheads when handling attack traffic with long sequences of out-of-order packets. Besides, with the increasing popularity of virtualization technologies, network intrusion detection on virtualized platforms is also proven to be useful in reducing overheads, as it supports dynamic scaling of computational resources and flexible deployment of detection functions. For example, in [12], J. Deng *et al.* built a virtualized IDS that is regulated by a virtualized controller for semantic consistency, correct flow update, buffer overflow avoidance, and optimal scaling in real-time. *vNIDS* [113] employed a detection state sharing mechanism to reduce the virtualization overhead of IDS, therefore, it achieves elasticity in detecting attacks with various profiles and also guarantees acceptable scalability.

2.4.3 Fine-grained Flow Statistic-based Detection

Both rule-based and signature-based detection systems barely maintain flow-level telemetry for the protected networks. Therefore, they are facing challenges in providing fine-grained statistics to precisely detect external attackers and malicious flows in distributed attacks. Many researchers have proposed methods to address the problem by exploring graph data structure for network anomaly detection using flow-level statistics so that they can achieve precise attack mitigation (*i.e.,* on only malicious flows) without causing collateral damage.

Scalability Issues

However, fine-grained flow statistics of a large network may become excessively massive and hard to be analyzed for distributed attack detection practically. To this end, many research efforts have been made by developing lightweight data structures that maintain flow statistics. *Kronecker graph* [114] is designed to model network flows using graphs generated by a non-standard matrix operation named Kronecker product, which is both realistic and practical. The authors of [115] leverage distributed in-memory graphs to maintain flow statistics by different computing nodes that collectively detect DDoS attacks cost-effectively. Many prior works take advantage of streaming (or online) algorithms to achieve attack detection with low computational cost. *STONE* [116] maintains flooding-related attributes for each protected group through continuous queries that merge current statistics with the past ones without creating new data points. The authors of [117] systematically reviewed processing methods for streaming graphs, such as 'insert-only graphs', 'graph sketches', and 'sliding window' that are helpful to reduce the computational costs in processing flow statistics. In [118], the authors developed an adaptive online classification scheme that detects network attacks on aggregated flow statistics.

Extracting Important Features

Extracting important and useful features from flow statistics for attack detection is another popular research direction. Principal component analysis (PCA) is a method to identify and extract important features in a classification task. The authors of [119] developed an augmented PCA model using Leibler divergence that outperforms the legacy methods in detecting anomalous network flows. To reduce redundant and irrelevant attributes, the authors in [120] proposed a multi-stage feature selection method that utilizes filters and stepwise regression wrappers. 41 commonly used features for network anomaly detection are examined and only 16 of them are proven to be effective in attack detection. The work in [121] introduced five groups of descriptive features (such as flow metadata features, sequence packets features, and general statistical features) of network flows. The supremacy of those attributes is demonstrated in detecting seven types of network attacks including SSH patator, DDoS, and port scan.

Statistical Methods using Flow Statistics

Besides, developing novel (statistical) methods using flow characteristics for better attack detection has also been explored by the community. For example, S. Jin *et al.* discuss their work in detecting SYN flooding attacks using a covariance analysis model in [122]. They have shown that the model can effectively distinguish benign and malicious flows by profiling their TCP headers. K. Lee *et al.* [123] applied clustering algorithms on a set of parameters that describe traffic features such as randomness of source and destination IP to differentiate DDoS traffic from normal communications. The authors in [124] employ a statistical metric named total variance distance that quantitatively measures the similarity between flows, which achieves better performance in detecting attack traffic than legacy methods.

2.4.4 Highlights

In this section, I classify existing attack detection methods into three types including proprietary rule-based, community signature-based, and fine-grained flow statistic-based detection.

Rule-based detection is widely adopted by the current industry for its scalability in deployment and operation. However, due to the increased agility and dynamicity of attack surfaces, such static methods are becoming less effective in precisely isolating malicious entities from the benign ones and insufficient to protect enterprise hosts that are larger in quantity and diverse in functionalities by manual configurations.

Signature-based detection often relies on input (*i.e.,* signature files) from open-source communities. Selecting proper and updated signatures developed by security experts can help IT departments quickly respond to emerging threat types. However, such open-source signatures may not be specifically designed for enterprise thus introduce compatibility issues. Besides, they are often packaged as software tools that provide higher management flexibility than proprietary solutions but also make them expensive to scale for a large network.

Attack detection leveraging fine-grained flow statistics have shown their supremacy in precisely identifying victims, attackers, and malicious flows of a distributed attack. However, maintaining and processing fine-grained statistics for the massive number of concurrent flows going through a large enterprise is not scalable especially in real-time. Therefore, achieving scalability while not compromising the fine-grained visibility of flow statistic-based methods is the key challenge to address before they could be widely deployed in practice.

2.5 Surveying Opportunities for Network Security by Emerging Paradigms

The advances of programmable networks and machine learning (ML) techniques bring new opportunities to address current problems in enterprise network security. In recent years, researchers have applied the two techniques in other aspects of network security, such as developing orchestration systems for flexible attack detection in ISP networks or proposing accurate algorithms to detect attacks of a certain type. Those prior works provide lessons for us to develop practical and effective security systems for a large enterprise network. In what follows, I discuss related research on network security using programmable networks and ML, respectively.

2.5.1 Using Programmable Networks for Network Security

The concept of programmable networks is the collection of techniques that make network functions flexible through network function virtualization (NFV) [125] and software-defined networking (SDN) [126]. Such technologies enable the run-time configuration and responsive update of flow rules and network functions. Thus, IT departments can flexibly customize security functions and orchestrate defense utilities to cope with their dynamic network and threat surfaces.

Practical Challenges

Although programmable networks hold promise in improving the current security ecosystem, several practical challenges slow its adoption [127], such as performance bottlenecks of software controllers, potential security issues of programmable networks, scalability issues of software-based network functions, and incompatibility with existing networked systems.

To this end, many efforts have been made to address some of the above problems. For example, R. Sommer *et al.* [128] proposed an NFV architecture that can fully utilize multi-core processors to achieve scalable network intrusion detection. *O3FA* [112] is presented as a lightweight engine for finite automata-based deep packet inspection that matches packets without reassembling flows. Thus, the system requires less buffer space compared with its counterparts. *StateAlyzr* [129] identifies and reduces the unnecessary operation processes for state clones in security middleboxes to achieve low computational overheads. *NetBricks* [130] employs a zero-copy software isolation mechanism that significantly reduces the performance overheads of typical NFV platforms. *OpenNetVM* [131] is developed as an NFV framework with high-level abstractions so that users can deploy their customized network functions without complex configurations. *ParaBox* [132] is designed as a hybrid packet processing pipeline that distributes incoming traffic into parallel network functions that outperform existing serial function chaining mechanisms. *StatelessNF* [133] separated the architecture of legacy virtual network functions into a state management component (*i.e.,* data store layer) and a stateless packet processing component which are orchestrated by SDN utilities so that the in-line traffic is processed in a more scalable manner. *vNIDS* [113] addresses ineffective detection and expensive provisioning of SDN/VNF-based systems by state sharing between detection modules and dynamic slicing of detection logic programs.

Current Research Prototypes

Apart from the research works aiming to address practical challenges, many prototypes have been developed to address various real-world attack detection problems using programmable networks.

For example, R. Braga *et al.* [134] developed a system using programmable switches to extract flow statistics for flooding attack detection. S. Lim *et al.* discussed their work in [135] that utilizes OpenFlow-based programmable switches to achieve flexible isolation of bots in DDoS attacks. In [136], K. Giotis *et al.* developed their system via a combination of OpenFlow and sFlow utilities to collect and process network statistics for scalable anomaly detection. *FlowTags* [137] uses SDN architecture to achieve flexible security enforcement through network-wide middleboxes that impose low computational overheads. *Bohatei* [11] uses SDN proactive and reactive flow rules to dynamically orchestrate network traffic forwarding at ISP-level so that attacks could be handled by the optimal security middleboxes in real time. C. Yoon *et al.* [138] demonstrated the feasibility of using programmable networks for cyber security by prototyping SDN-based in-line firewalls, passive IDS, network anomaly detectors, and specialized security functions. *Atlantic* [139] leverages the flexibility of SDN to jointly detect, classify, and mitigate malicious flows in a small-sized network of 100 hosts and two switches. The authors of [140] use SDN reactive routing to only forward the first several packets of each network flows for deep packet inspections. J. Deng *et al.* [12] built a virtual firewall architecture using SDN and NFV to achieve elastic rule placement and flexible detection functionalities. *Sonata* [86] achieves scalable traffic processing by offloading heavy and repetitive network functions from software processors to hardware programmable switches.

2.5.2 Using Machine Learning Techniques for Network Security

Machine learning techniques have proven their usefulness in making accurate classification and precise detection under various scenarios such as image processing and speech recognition. While the maturity of ML applications in network security is still in its early stage of industrial adoption [13, 141], researchers have developed many ML approaches for a large range of topics in cybersecurity [142] that provide insightful lessons for the community.

For example, *MADAM ID* proposed in [143] is a framework that uses ML-based data mining to process network audit data for intrusion detection. J. Shum *et al.* [144] leveraged simple neural networks to detect Internet attacks. *BotMiner* [40] employs clustering algorithms to identify botnet groups by analyzing network traffic patterns without a priori knowledge of their behavioral signatures. L. Koc *et al.* [145] proposed a Hidden Naive Bayes (HNB) method for network intrusion detection that outperforms other machine learning models that are either not effective for handling high dimensional data, identifying dependent features, or incurring high computational overheads. M. Javed *et al.* [146] designed a scheme that specifically detects SSH brute-forcing using a beta-binomial distribution model. The authors in [147] developed an ensemble model that uses Bayesian Network with Gain Ratio for feature selection Artificial Neural Network for attack detection. C. Hsieh *et al.* [148] proposed a DDoS attack detection system using neural networks on Apache

Spark big data computing clusters, which can handle traffic with large volumes. *DeepLog* [149] leverages deep learning algorithms for anomaly detection on system logs collected from enterprise hosts. H. Siadati *et al.* [150] used ML-based anomaly detection algorithms to identify anomalous logins within an enterprise network. D. Tang *et al.* [151] developed data-driven models to detect low-rate DoS attacks that have abnormal patterns in frequency, fluctuation, and distribution of TCP flows. I note that most of the existing works focus on developing high-accuracy models and algorithms to detect a certain type of attacks, which prove the effectiveness of ML-based methods in solving network security problems and pave the way for developing data-driven solutions in asset management and distributed attack detection specifically and practically for large enterprises.

2.5.3 Highlights

In this section, I have discussed the advances and adoptions of programmable networking and ML techniques in the area of network security.

Programmable networking paradigms support flexible and real-time traffic routing and processing compared with legacy systems. Although there are still many practical problems to solve, ranging from the scalability issues due to limited software and hardware resources (*e.g.,* data-plane switch memory and flow entries), the difficulty for administrators to acquire related skills, and compatibility with existing network infrastructures. The techniques enable IT departments to adjust the visibility and granularity of network telemetry for security inference in real-time so that precise fine-grained statistics can be collected based on the status of networks and the evolution of attacks.

ML and its associated data-driven techniques give promises in classifying traffic patterns of hosts and detecting attacks through automatic models developed from network big data instead of manually defined thresholds and signatures. However, applying ML algorithms in network security is still in its early stage due to many existing barriers [13], such as balancing attribute descriptiveness and scalability, handling false positives that may lead to serious security implications and explainability of detection results by performant models. Carefully addressing the above problems is a prerequisite for the success of an ML-based solution in network security.

2.6 Related Surveys on Network Security

I now discuss related survey articles that focus on different aspects of network security.

Categorizing attack detection methods: A group of literature reviews focuses on the categorization of detection methods that have the potential to handle network attacks. J. Mirkovic *et al.* [14] systematically introduce the taxonomy of DDoS and their corresponding defense mechanism. The authors in [141] categorized the architectural types of network intrusion detection systems that use different underlying techniques. S. T. Zargar *et al.* [78] highlighted DDoS defense mechanisms categorized by their deployment location and the action time in which defense takes place. The authors in [152] comprehensively discussed vulnerabilities existing in the networking ecosystem that are targeted by emerging cyber-attacks and their countermeasures. The types and mechanisms of data mining and machine learning methods and their applications in cyber-security research have been discussed in [142]. Flow-based intrusion detection techniques, datasets, and prototypes are summarized in [153]. A. Voronkov *et al.* [56] provided a systematic review on the usability aspect of firewall configurations. A. Tundis *et al.* [23] reviewed existing tools that scan network vulnerabilities for benign or malicious purposes. The authors in [154] discussed functionalities of popular next-generation firewalls (NGFW) and their capabilities in coping with emerging network threats. C. Chen *et al.* [155] focused on the architecture of situational awareness systems for network security, which include data collection, situational understanding, prediction, and visualization. S. Sengupta *et al.* [156] comprehensively discussed the effective methods to defend against attacks originated from moving targets.

Attacks on certain network types: There is also a cluster of literature reviews focused on attacks on emerging network types that hold different characteristics and vulnerabilities compared with legacy networks. X. Chen *et al.* [157] systematically discussed security problems in wireless sensor networks and evaluate the effectiveness of existing defense techniques. The authors in [158] and [159] discussed security issues and requirements for an effective defense architecture in the era of software-defined networks. The works in [106], [107], and [160] focused on network intrusion detection and DDoS detection for IoT networks. N. Agrawal *et al.* [16] particularly focus on the defense mechanisms against DDoS attacks for cloud computing networks. J. Cao *et al.* [161] summarized the security challenges introduced by various aspects of 3GPP 5G networks and their security requirements which are not addressed by current solutions. A. Bhardwaj *et al.* [162] surveyed scientific and industrial solutions in DDoS detection for cloud networks.

The focus of this survey: To the best of my knowledge, prior surveys categorized certain type of attacks (*i.e.,* DDoS) and their defenses by the broad aspects such as target locations, attacking mechanisms, and exploited network vulnerabilities. Instead, this survey narrowly focuses on the distributed (volumetric) network attacks, their countermeasures, and opportunities by emerging paradigms that are specifically applicable for enterprise networks. Therefore, this survey provides an important reference for the further research on enterprise network security which has not been fully covered by prior surveys.

2.7 Discussion on Research Gaps

After an extensive review of current techniques in asset behavior monitoring and distributed attack detection for enterprise networks, I now articulate open issues that are yet to be solved by future research.

Dynamic and scalable host monitoring: The large complexity of networked assets and dynamicity of their communication patterns are hard to be fully captured by legacy methods through static configurations. Dynamic networked graphs are effective in covering host behavioral profiles but infeasible to be maintained for an entire large enterprise. Therefore, developing methods to achieve scalable monitoring of assets while guaranteeing dynamic fine-grained visibility into necessary traffic fractions is a valuable research direction. In Chapter 4 and 6), I show my efforts in achieving this objective for host monitoring and attack detection using the programmable networks technique, which enables us to reactively collect fine-grained telemetries when necessary.

Role-aware attack detection: Different types of networked assets exhibit distinguishable communication patterns and vulnerabilities that make them potential targets of distributed attacks. Detection mechanisms enforced for each enterprise host are expected to be properly customized according to their roles and vulnerabilities. Current attack detection methods either leave this job to operators who configure policies on certain IP addresses or simply apply a generic detection mechanism to the entire network. Therefore, smart configurations of attack detection mechanisms based on the traffic profile of enterprise hosts would be a significant contribution. In Chapter 3 and 4, I will discuss my proposed systems that continuously classify and monitor the roles of enterprise hosts to help IT departments locate their internal network vulnerabilities and configure proper security enforcement accordingly.

Explainability of ML-based methods: Although ML methods have proven their high accuracy in host classifications and attack detection, due to the limitation of training datasets, imperfection of statistical features, and biases from algorithms, such methods may not perform well in operational networks. Therefore, the results from ML-based systems have to be explainable so that IT departments can understand the reason for each classification/detection to avoid mishandling of false-positives. In Chapter 3,4, and 5, I develop explainable ML-based methods for host classification and attack detection by conducting systematic network big data analysis to extract descriptive attributes, developing post hoc reasoning for each classification result, and performing fine-grained inspections on the detected anomalous hosts/attacks for further verification.

Self-driving enterprise security: Current enterprise network security and management systems often require configurations from IT departments to specify their protected devices, protection types (*e.g.,* scans or DDoS), detection thresholds, and mitigation mechanisms which may become quite complex for a large organization. Manually maintaining such a large volume and sophisticated policies may lead to improper configurations such as inconsistencies and conflicts. To reduce the potential risks introduced by human errors, it is worthwhile to develop self-driving security systems that can be operated in a complex enterprise network with minimum configuration. Driven by this idea, in the rest of this thesis (*i.e.,* Chapter 3, 4, 5, and 6), I develop my systems to classify enterprise assets and detect distributed attacks in an automatic manner without requiring complex configurations to specify inference mechanisms and the protected IP lists.

2.8 Conclusion

This survey focuses on the distributed network attacks on enterprise assets and countermeasures including asset monitoring and attack detection systems. I discuss the diversity of distributed attack (reconnaissance and DDoS) on enterprise assets, review existing industrial and research methods in monitoring network profiles of enterprise hosts and detecting distributed attacks, and highlight two emerging techniques (*i.e.,* programmable networks and ML) that bring new opportunities in addressing enterprise network security problems. Lastly, I highlight several research gaps as valuable directions that are worthwhile to be explored.

Chapter 3

Analyzing Enterprise DNS Traffic to Classify Assets and Track Cyber-Health

Contents

The Domain Name System (DNS) is a critical service that enables domain names to be converted to IP addresses (or vice versa); consequently, it is generally permitted through enterprise security systems (*e.g.,* firewalls) without restriction. This has exposed organizational networks to DDoS, exfiltration, and reflection attacks, inflicting significant financial and reputational damage. Large organizations with loosely federated IT departments (*e.g.,* Universities and Research Institutes) often do not even know all their DNS assets, let alone the attack surface they expose to the outside world.

In this chapter, I address the "DNS blind spot" by developing methods to passively analyze live DNS traffic, identify organizational DNS assets, and monitor their health on a continuous basis. My first contribution performs a comprehensive analysis of all DNS traffic in two large organizations (a University Campus and a Government Research Institute) for over a month, and identifies key behavioral profiles for various asset types such as recursive resolvers, authoritative name servers, and mixed DNS servers. For my second contribution, I develop an unsupervised ML model that classifies enterprise DNS assets using the behavioral attributes identified, and demonstrate that my method successfully classifies over 100 DNS assets across the two organizations. My final contribution continuously tracks various health metrics across the organizational DNS assets and identifies several instances of improper configuration, data exfiltration, DDoS, and reflection attacks. I believe the passive analysis methods in this chapter can help enterprises monitor organizational DNS health in an automated and risk-free manner.

3.1 Introduction

Enterprise networks are large in size with many thousands of connected devices and dynamic in nature as hosts come and go, and servers get commissioned and decommissioned to adapt to the organization's changing needs. Enterprise IT departments track such assets manually today, with records maintained in spreadsheets and configuration files (DHCP, DNS, Firewalls, etc.). This is not only cumbersome, but also error-prone and almost impossible to keep up-to-date. It is, therefore, not surprising that many enterprise network administrators are not fully aware of their internal assets [163], and consequently do not know the attack surface they expose to the outside world. The problem is even more acute in University and Research Institute campus networks for several reasons [164]: (a) they host a wide variety of sensitive and lucrative data, including intellectual property, cutting-edge research datasets, social security numbers, and financial information; (b) their open-access culture, decentralized departmental-level control, as well as federated access to data makes them particularly vulnerable targets for unauthorized access, unsafe Internet usage, and malware; and (c) they typically have a high-speed network infrastructure that makes them an attractive launchpad for volumetric attacks on other entities.

DNS is a protocol of choice exploited by cyber-criminals and botnets as it can readily bypass firewalls and security middleboxes. Due to the open nature of DNS, it is common for organizations to apply few (if any) restrictions (*e.g.,* firewall rules) to DNS traffic. Thus, it is unsurprising to see the increasing frequency and quantity of malware compromised devices and attacks that leverage DNS protocol [165–169], such as DDoS attack, DNS tunneling and sensitive data exfiltration.

Enterprises typically host various kinds of DNS assets. They will typically host a small number of recursive resolvers that proxy DNS requests from internal hosts to external DNS servers, and also cache results to reduce the number of external queries. Individual hosts may choose to over-ride the enterprise recursive resolvers, such as by manually changing their preferred resolver to a public one (such as Google's 8.8.8.8 and CloudFlare's 1.1.1.1), but in general, a majority of hosts will use the default recursive resolver provided by their organization. In addition, enterprises typically host a number of authoritative name servers to serve the various domains belonging to the organization. For example, organization-wide services (like email, VPN, etc.) may be managed by central IT. At the same time, each department may operate its own authoritative name server to resolve department-specific web pages. It is not uncommon for the various IT entities to operate in silos, often unaware of the assets being managed by the other. To make matters worse, on-campus retail stores (bookshops, food outlets, etc.) that lease connectivity from the campus may also be housing their own DNS assets, which are often poorly secured as they lack the skills.

Existing enterprise network security appliances such as border firewalls and intrusion detection systems do not provide fine-grained visibility of internal DNS-related hosts and assets, creating significant "blind spots" for the IT department [170]. While there is a significant body of existing academic research on DNS traffic analysis and DNS security, existing works either focus on forensic analysis of logs collected from DNS servers, such as recursive resolvers on the Internet [171–174] and domain registrars [175], or concentrate on packet-level domain names [176–179]. This work is the first to develop data-driven methods to automatically map and monitor DNS assets in an enterprise. Commercial security appliances can consume insights obtained from my methods. My contributions are three-fold:

First, in §3.3, I perform a comprehensive forensic analysis of DNS traffic from two large organizations collected over 32 days, comprising nearly a billion queries/responses. I examine their network properties (IPv4/v6, UDP/TCP, etc.), functional properties (unpaired queries/responses, errors, etc.), and service properties (lookup types, record types, etc.). These enable us to build behavioral profiles of how various DNS assets (recursive resolvers, authoritative name servers, and mixed servers) behave.

Second, in §3.4, I use the behavioral profiles learned above to identify key attributes of each type of DNS asset, extract such attributes efficiently in real-time, and develop an unsupervised machine learning model using clustering algorithms to dynamically and continuously classify asset types, including recursive resolvers, authoritative name-servers, mixed DNS servers, and regular end-host clients that may or may not be subject to enterprise network address translation (NAT). I apply my method to identify over 100 different DNS assets across the two organizations, and validate my results by cross-checking with IT staff. My method further identifies assets that were commissioned/decommissioned or changed during the monitoring period, further validating its utility in dynamically changing environments.

Third, in §3.5, I develop data-driven metrics that commercial SIEM[1] platforms can consume to track the cyber health of DNS assets or identify their anomalous behaviors – the metrics are inspired by the insights obtained from §3.3. My methods reveal a prevalence of poor server configurations in both organizations, allowing attackers to exploit them for reflection attacks. Further, my methods are able to identify the organizational DNS assets that are complicit in scans, DDoS, and data exfiltration. I give proposals on how these DNS threats can be mitigated.

Taken together, my contributions help enterprises address their current blind spot in monitoring their DNS assets and the threats they are exposed to. The passive analysis I propose is automated, risk-free, and particularly beneficial to large organizations with numerous assets managed by diverse personnel.

3.2 Related Work

I now discuss related works on DNS traffic analysis (§3.2.1) and DNS attacks (§3.2.2) with highlights on the novelty of my work.

[1]Security Information and Event Management.

3.2.1 Analysis of DNS Traffic

DNS traffic has been analyzed for various purposes, ranging from measuring performance (effect of Time-to-Live of DNS records) [173, 178, 180] to identifying malicious domains [175, 177, 181] and the security of DNS [182–185]. In this chapter I have profiled the pattern of DNS traffic for individual hosts of two enterprise networks to map DNS assets to their function and thereby identify their relative importance and health for efficient monitoring and security.

DNS data can be collected from different locations (such as from log files of recursive resolvers [173, 186] or authoritative name servers) or with different granularity (such as query/response logs or aggregated records). Datasets used in [182–184] contain DNS traffic for top level domains such as `.com`, and `.net`. The work in [187] studies the root cause of query failures by analyzing DNS logs collected from recursive resolvers operated by three Internet service providers. I collect my data at the edge of an enterprise network, specifically outside the firewall at the point of interconnect with the external Internet. I note that while using data from resolver logs can provide detailed information about end hosts and their query types/patterns, this approach limits visibility and may not be comprehensive enough to accurately establish patterns related to the assets of the entire network.

Considering studies related to domain names, [181] inspects DNS traffic from top-level domain servers to detect abnormal activity and identifies key characteristics of malicious domains in terms of their resource records and lookup patterns, PREDATOR [175] derives domain reputation using registration features to enable early detection of potentially malicious DNS domains without capturing traffic, and [188] gives practical recommendations for using public domain ranking lists in security research, based on their temporary changes. As for detection of such suspicious domain names, [171] investigates into the coexistence of names in distributed DNS recursive resolvers, [189] explores the value of game theory in detecting malicious domain names generated by hidden Markov models and probabilistic context-free grammars, which can bypass legacy detection methods.

Key novelty of this work: Prior works analyzed DNS traffic collected from different vantage points with various objectives. This paper measures traffic at the edge of an enterprise network. I am the first to profile the behavior and health of enterprise hosts by identifying patterns in DNS communications and distribution of various DNS packets. I highlight some interesting observations like when benign query names like `google.com` are misused in cyber-attacks (scans and query-floods) targeting enterprise networks and services.

3.2.2 Studies on DNS Attacks

From the aspect of DNS attacks, DNSSEC [190] has been proposed for more than a decade to deprecate information integrity attacks such as cache poisoning [191], however, the authors of [182] study the adoption of DNSSEC, highlighting that only 1% of domains have implemented this secure protocol due to difficulties in the registration process and operational challenges – it is also verified from my results of two enterprises that very little fraction of DNS traffic are mapped to DNSSEC. The introduction of DNSSEC brings more potentials for volumetric attacks, and some researchers [185] have reported that the amplification factor of DNSSEC is quite high (*i.e.,* up to 44 to 55) whereas this measure is 6 to 12 for regular DNS servers. Besides, [183, 184] focus on authoritative name servers used as reflectors in DNS amplification attacks – it indicates the potential vulnerabilities of enterprise DNS servers to be mis-used in DDoS attacks. Work in [5] proposed a hierarchical graph structure with anomaly detection models to identify distributed DNS attackers outside an enterprise network at various levels of aggregation (*e.g.,* host, subnet, and AS).

Key novelty of this work: Existing works focus on highlighting certain vulnerabilities of the DNS protocol or developing methods for detecting DNS attacks. My work, instead, systematically profile and track the DNS-related behavior of connected hosts in an enterprise. The system I develop and insights I draw will help IT departments better map their assets, discover potential DNS vulnerabilities, and detect misbehaved (potentially infected) hosts on their network.

3.3 Analysis of DNS Traffic from Two Enterprises

In this section, I analyze the characteristics of DNS traffic collected from the border of two enterprise networks, a large University campus (*i.e.,* the University of New South Wales (UNSW)) and a national research institute (*i.e.,* Commonwealth Scientific and Industrial Research Organisation (CSIRO)). I start by introducing my measurement setup is described in §3.3.1. In §3.3.2, I discuss the "network", "functional", and "service" properties of one-week DNS packets collected from both organizations to highlight their normal and abnormal profiles. I then (in §3.3.3) focus on the distribution of DNS packets among each enterprise host to reveal their DNS behavioral patterns and unhealthy traffic compositions.

3.3.1 Measurement Setup

In both organizations, the corresponding IT department provisioned a full mirror (both inbound and outbound) of their Internet traffic (each on a 10 Gbps interface) to my data collection system, shown in Fig. 3.1, from their border routers (**outside** of the firewall), and I obtained appropriate ethics clearances for this study[2]. I extracted DNS packets from each of the enterprise Internet traffic streams in real-time by configuring rules for incoming/outgoing IPv4 and IPv6 UDP packets for port 53 on a programmable network switch. It is worth noting that a tiny fraction of DNS

[2]UNSW Human Research Ethics Advisory Panel approval number HC17499, and CSIRO Data61 Ethics approval number 115/17.

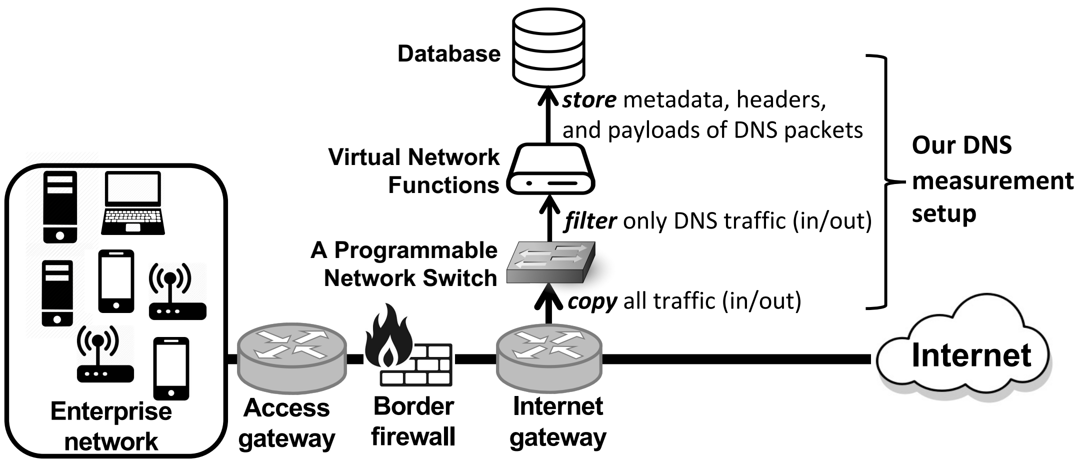

Database
store metadata, headers, and payloads of DNS packets
Virtual Network Functions
filter only DNS traffic (in/out)
A Programmable Network Switch
copy all traffic (in/out)
Our DNS measurement setup
Enterprise network
Access gateway
Border firewall
Internet gateway
Internet

Figure 3.1: My DNS measurement setup.

lookups might be carried by TLS [192] and HTTPS [193] that are beyond the scope of this chapter. The mirrored DNS traffic was processed by a virtual network function running on a generic server (with DPDK [194]) which parses headers (network, transport, application) and payload of each DNS packet, and stores them into my database. The study in this chapter considers the data collected over a month period of 3 June to 4 July 2019 (*i.e.,* beginning of an academic term in the university). In this section, I focus on the analytic results for 1 week worth of DNS traffic capture from both organizations during 3 June to 9 June 2019.

3.3.2 Understanding DNS Traffic at Enterprise Network Border

I begin by examining "network", "functional", and "service" properties of DNS packets, which provide answers for the following three questions related to DNS traffic profiles of an organization. How does each DNS packet get carried at network-level? Is each DNS packet with correct or error functionality? What is the service type of each DNS packet?

Table 3.1: Network properties of DNS packets in my dataset.

			Incoming			Outgoing		
			query	**response**	**malformed**	**query**	**response**	**malformed**
University	**IPv4**	**TCP**	258, 315	217, 210	**298,979 (38%)**	244, 633	553, 097	4, 824(0.3%)
		UDP	166, 492, 688	181, 610, 373	**56,665,050 (14%)**	190, 974, 279	38, 321, 129	2, 158, 933(0.9%)
	IPv6	**TCP**	1, 223	23, 080	5, 261(17%)	25, 525	1, 203	38(0.1%)
		UDP	10, 989, 944	53, 592, 304	200, 323(0.3%)	54, 673, 191	7, 182, 025	207(0.0006%)
Rsrch. Ins.	**IPv4**	**TCP**	25, 829	175, 786	18, 542(8%)	200, 269	28, 421	3, 375(1.4%)
		UDP	48, 629, 262	53, 423, 998	1, 034, 531(1.0%)	59, 638, 578	22, 344, 154	2, 493(0.003%)
	IPv6	**TCP**	425	11, 445	14, 394(55%)	19, 708	338	274(1.3%)
		UDP	5, 889, 648	14, 068, 272	82, 050(0.4%)	16, 455, 764	6, 502, 566	224(0.0009%)

Network Property

DNS packets can be carried by either TCP or UDP at the transport layer via IPv4 or IPv6 protocols. Table 3.1 summarizes the composition of DNS packets (in my dataset) by their network properties.

Unsurprisingly, the majority of DNS packets are carried by IPv4 protocol, and it is clear that the adoption of IPv6 in DNS communications has become non-negligible in both organizations. I found that 21.03% and 21.84% of outgoing DNS packets in the university and research networks, respectively, are IPv6, while this measure for incoming DNS traffic of the two organizations is 13.78% and 16.26%.

Considering the transport layer, DNS over UDP seems to be default for enterprise hosts, accounting for more than 99% of outgoing and incoming packets in both organizations, while DNS over TCP is still staying minority (less than 0.3%). I note that DNS occasionally uses TCP when the size of the request or the response is greater than a single packet such as with responses that have many records or many IPv6 responses.

Focusing on the correlation between DNS queries and responses, I highlight four pairs of query/response in Table 3.1, as examples – each pair is color-coded for identification. It can be seen that the number of outgoing queries is slightly higher than the number of incoming responses, suggesting unanswered DNS lookups made by enterprise hosts (green and purple pairs in Table 3.1). I also observe that count of incoming queries over IPv4 UDP is more than double the count of outgoing responses in both organizations (*e.g.,* red and yellow pairs in Table 3.1), highlighting the prevalence of DNS scans and floods on enterprise networks. However, this is not substantiated in IPv6 packets. Lastly, I note that in the university network, the count of outgoing TCP-based responses over IPv4 is more than double the number of their corresponding queries (the gray pair in Table 3.1), indicating non-negligible malicious unsolicited IPv4 TCP responses generated by the internal university hosts (*e.g.,* involved in DNS reflection attacks).

I found that 9.82% and 0.54% of total DNS packets in the university and research institute dataset, respectively, are malformed. These packets cannot be correctly parsed as their DNS header information mismatched the payload content. There are various reasons for having malformed DNS packets mentioned in [14, 195, 196] such as malicious traffic crafted by attackers and packet truncation or distortion during transmission. It can be seen that there are more malformed incoming packets compared to outgoing packets, as highlighted by percentage values (computed per row per direction) under malformed columns in Table 3.1.

Another observation is that malformed DNS packets are more likely carried over TCP. For example, an inbound packet over IPv4 TCP in the university network is malformed with a probability of 38%, while that is 14% over IPv4 UDP (bold text in Table 3.1). Besides, when comparing the two organizations, I observed that the university network sends more malformed packets in total fraction than the research institution, particularly for outbound IPv4 UDP packets (0.9% versus 0.003% for the university and the research institute, respectively). It indicates frequent malicious activities originated from university hosts, as the university network is open and less restricted, while the research institute does not allow BYOT (bring-you-own-technology) devices and has strict enforcement for network security.

Table 3.2: Functional properties of DNS packets.

		Incoming		Outgoing	
		IPv4	IPv6	IPv4	IPv6
University	Unanswered qry.	130,683,135	3,813,677	11,431,123	1,105,818
	Unsolicited resp.	2,039,794	22,486	2,806,358	5,738
	NameError pairs	7,493,599	1,885,742	5,164,713	1,532,410
	Serv.Failure pairs	3,897,549	34,643	1,363,391	112,618
	Qry.Refused pairs	24,820,580	16,409,541	2,102,724	26,130
	OtherError pairs	113,291	90	794	0
	Non-enterprise pairs	9,153,748	252,860	178,234,417	53,096,280
	Enterprise pairs	26,914,120	6,924,630	1,553,372	496,618
Research Institute	Unanswered qry.	29,159,158	182,886	9,912,604	2,843,892
	Unsolicited resp.	3,673,541	448,137	2,876,642	795,717
	NameError pairs	2,730,158	974,480	3,775,508	1,011,591
	Serv.Failure pairs	248,275	19,715	2,389,070	5,601
	Qry.Refused pairs	1,390,138	245,390	781,259	133,599
	OtherError pairs	17,061	50	621	230
	Non-enterprise pairs	2,071,310	558,205	48,754,035	13,275,249
	Enterprise pairs	17,424,623	5,148,982	1,172,208	356,331

Functional Property

In terms of functional property, I categorize DNS packets into three clusters: (a) unpaired packets (*i.e.,* queries with no reply or responses without a corresponding query), (b) DNS lookups with a reply containing errors, and (c) successful DNS lookups.

Unpaired packets: This category is captured by two rows labeled as "*unanswered qry.*" and "*unsolicted resp.*" in Table 3.2. Unanswered queries (highlighted by red cells in Table 3.2), carried over both IPv4 and IPv6, contribute to a large fraction of total incoming DNS packets – 40.4% and 30.8% in the university and research institute, respectively – this is due to frequent DNS scans and query floods targeting enterprise DNS infrastructure. On the other hand, unanswered outgoing queries only account for a relatively smaller fraction in each organization (*i.e.,* 2.4% and 7.9%). Moving to unsolicited responses, their fraction in both inbound and outbound traffic are quite similar. This is mainly because of packet drop during transmission, misconfiguration of external DNS servers or DNS-based reflection attacks from/to the enterprises [197],[198].

DNS lookups with error reply: Now I focus on the DNS lookups with error replies, classified by their `errorCode` in the headers of response packets. Top 3 popular error types in both enterprises are listed as `NameError`, `ServerFailure` and `QueryRefused` in Table 3.2. Other minorities are combined as `OtherError`. `NameError`, also known as `Non-Existence Domain`, are triggered if the requested domain name is not correct. It might be because of typo errors from legitimate users or malicious queries sent from malware-infected hosts as reported in [172, 179]. `ServerFailure` and `QueryRefused` indicate that target DNS servers are not able to provide resolved answers for various reasons such as zone restrictions or incorrect query formats. `NameError` is the most popular reason for error responses of inbound and outbound DNS lookups in the two enterprises, except for inbound traffic of university network, whose top error reason is `QueryRefused`.

Successful DNS lookups: Given a successful pair of DNS packets (*i.e.,* DNS query-response pair with `NoError` flag), their requested domain names can be classified as either relevant (*i.e.,* belong to services provided by the organization) or irrelevant. For organizations running their authoritative name servers for their domain names, I expect that all inbound DNS lookups are relevant to the enterprise. However, a non-negligible portion of inbound non-error DNS lookups asks for irrelevant domain names in both enterprises (21.7% and 10.4%, respectively). They are likely to be malicious DNS queries involved in scans or floods; furthermore, improperly configured DNS servers in the enterprise resolved those irrelevant questions. Outbound DNS lookups for two organizations contain a tiny portion of questions for the enterprise services (0.8% and 2.4%). As verified in my dataset, the top destinations of those queries are public recursive resolvers such as `8.8.8.8` and `8.8.4.4` operated by Google. It shows some hosts (for some reason) bypassed the local DNS caches, operational within each enterprise, by choosing public resolvers for their

Table 3.3: Service properties of DNS packets in my dataset.

		Incoming		**Outgoing**	
		IPv4	**IPv6**	**IPv4**	**IPv6**
University	**A pairs**	19,986,211	3,692,671	111,896,351	29,538,537
	AAAA pairs	7,782,897	2,014,541	32,223,426	6,615,980
	PTR pairs	2,927,101	594,635	24,749,068	15,775,549
	MX pairs	1,413,019	210,452	831,571	192,365
	SPF pairs	43,943	8,600	109	28
	TXT pairs	723,796	64,690	4,415,435	659,723
	CNAME pairs	79,235	23,408	11,708	1,693
	SRV pairs	599,022	197,023	2,678,513	103,494
	SOA pairs	220,711	88,752	714,524	299,316
	NS pairs	1,057,700	223,808	727,438	358,427
	ANY pairs	**1,205,822**	46,315	114,584	9,592
	Other pairs	21,990	10,305	1,209,553	3,754
Research Institute	**A pairs**	7,664,442	1,585,811	21,571,867	6,174,823
	AAAA pairs	2,287,039	755,134	23,774,650	6,107,818
	PTR pairs	7,677,620	2,998,552	2,040,424	599,030
	MX pairs	441,075	117,015	301,651	84,904
	SPF pairs	3,782	662	15,974	3,984
	TXT pairs	120,308	13,198	1,099,786	342,250
	CNAME pairs	43,399	5,556	19,933	116
	SRV pairs	230,046	33,483	364,039	79,366
	SOA pairs	222,796	40,515	0	0
	NS pairs	683,641	132,410	532,757	179,340
	ANY pairs	101,867	23,012	830	447
	Other pairs	18,054	988	1,512	165

DNS server.

Service Property

Successful DNS lookups are asking for various types of services, such as IPv4 address (A type), IPv6 address (AAAA type), and reverse lookup for domain names (PTR type). Statistics for success inbound/outbound lookup pairs are shown in table 3.3.

I start with the popularity of successful lookup types in both networks. It is clear that for both organizations, requests for IPv4/IPv6 addresses and reverse lookups for domain names are dominant types towards either inbound or outbound DNS traffic. The corresponding texts are marked as blue in Table 3.3. Similarly, significant amount of email-related (*i.e.,* `MX` and `SPF`), text exchange (`TXT`), DNS service-related (*i.e.,* `CNAME`, `NS` and `SOA`) and location (`SRV`) lookups are also observed in both organizations. Besides, no outbound DNS lookup is observed for `SOA` (that asking for authoritative information of a zone) in the research institute, while few outbound lookups for `SPF` (requesting authorized email servers of a domain) were observed in the university network.

I discovered the occurrence of DNS lookup types that are not suggested by the industry. Although the deprecation of `ANY` type requests has been announced by the networking community since 2019 [199][200] for their lack of legitimate purpose and abundant misuse in reflection attacks, I still find a large number of `ANY` type DNS lookups exist for the inbound traffic in both organizations, particularly for the university network (marked as bold red text). Focusing on the outbound traffic, university hosts sent out many such types of deprecated requests. In contrast, hosts in the research institute rarely had such activities (relevant cells are marked as red). Besides, many `A6` (deprecated version of lookups for IPv6 address) and `NAPTR` (mapping domain names to host URLs) are found in outbound requests in the university network, respectively contributing to 0.15% and 0.31% of the total count of outbound queries.

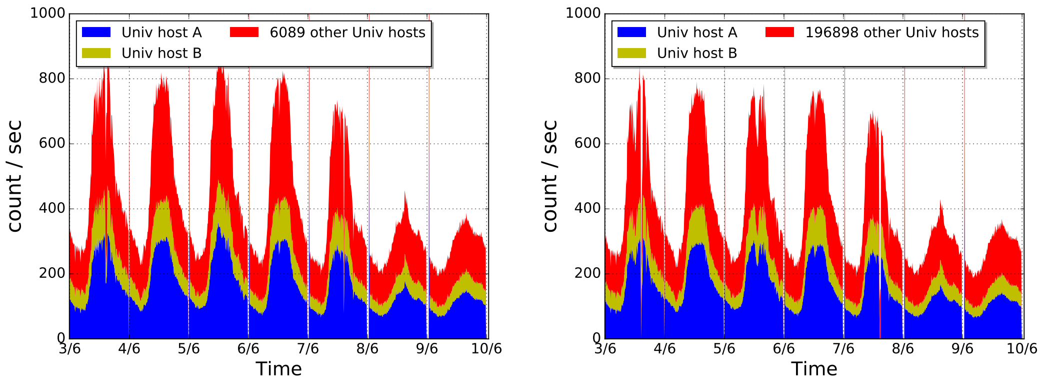
I now look at statistics of DNS lookups related to DNSSEC in both organizations. DNSSEC [190] has been proposed for more than a decade to strengthen information integrity of DNS data, prior measurement studies [182] on domain registrars resulted that the adoption of such extension is still in early stage. Fairly similar observations were made in both networks, as there are 0.005% inbound lookups and 0.1% outbound lookups are associated with DNSSEC services in the university campus, including `DNSKEY`, `DS`, `RRSIG`, `NSEC`, `NSEC3` and `DLV`. The fraction value of such inbound and outbound lookups for the research institute are 0.005% and 0.2%, respectively.

3.3.3 Profiling DNS Behaviors of Enterprise Hosts

Enterprises typically operate two types of DNS servers: (a) **recursive resolvers** are those that act on behalf of end-hosts to resolve the network address of a domain name and return the answer to the requesting end-host (recursive resolvers commonly keep a copy of positive responses in a local cache for time-to-live of the record to reduce frequent recursion), and (b) **authoritative servers** of a domain/zone are those that receive queries from anywhere on the Internet for the network address of a sub-domain within the zone for which they are authoritative (*e.g.,* `organizationXYZ.net`).

In order to better understand the DNS behavior of various hosts (and their role) inside an enterprise network, I divide the DNS dataset into two categories: (a) DNS queries from enterprise hosts that leave the network towards a server on the Internet along with DNS responses that enter the network, (b) DNS queries from external hosts that enter the network towards an enterprise host along with DNS responses that leave the network.

This analysis helps us identify important attributes related to host DNS behavior, characterizing its type/function, including authoritative name server, recursive resolver, or end-host inside the enterprise that may not always be fully visible to the network operators. This also enables us to capture the normal pattern of DNS activity for various hosts and identify the abnormal traffic status of DNS infrastructures.



(a) Outgoing DNS queries. (b) Incoming DNS responses.

Figure 3.2: University campus: outgoing queries and incoming responses, measured during 3 June to 9 June 2019.

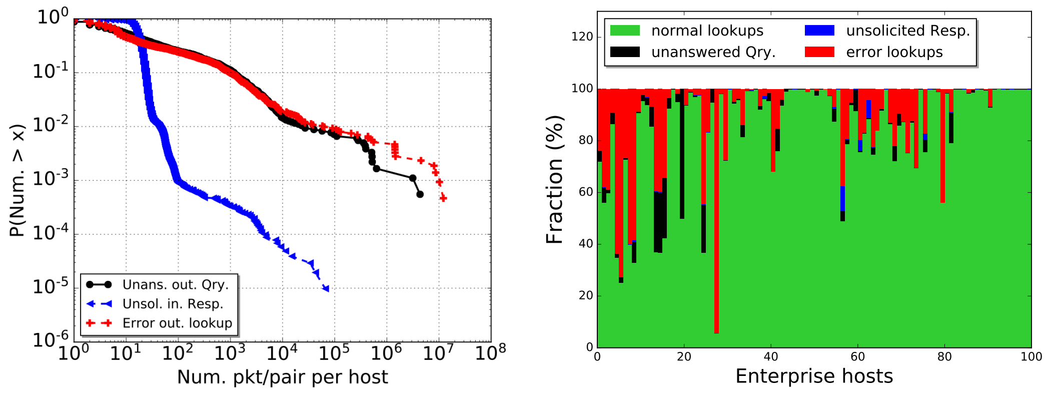
Outgoing Queries & Incoming Responses

Fig. 3.2 shows a time trace of DNS outgoing queries and incoming responses for the university campus[3], with granularity over 10-minute intervals on a typical semester week.

The university network handles on average 417 outgoing queries and 408 incoming responses per second. As discussed in Table 3.1, 4.9% of outgoing queries are "unanswered" (*i.e.*, 12.5M out of 256.2M) during the week. In addition, 2.06% of incoming responses to the university campus network (*i.e.*, 2.1M out of 99.9M) are "unsolicited" on the same day.

Query per host: I now consider individual hosts in each enterprise. Unsurprisingly, the majority of outgoing DNS queries are generated by only two hosts, A and B, in the network, *i.e.*, 66.8% of the total in the university campus (shown by blue and yellow shades in Figures 3.2(a)). These hosts are also the primary recipients of incoming DNS responses from the Internet. I have verified with the IT department of the enterprise that both hosts are primary recursive resolvers of this organization. In addition to these recursive resolvers, I observe a number of hosts shown by red

[3]I omit results for the research institute in this section, as fairly similar observations were made.



(a) CCDF: # Unwanted DNS pkts per host. (b) Traffic composition per host.

Figure 3.3: University campus: (a) CCDF of # unwanted (outgoing queries and incoming responses) DNS packets and (b) their total fractions per host, measured during 3 June to 9 June 2019.

shades in Fig. 3.2(a) that generate DNS queries outside of the enterprise network. The 6,089 other University hosts in Fig. 3.2(a) are either: end-hosts configured by public DNS resolvers that make direct queries out of the enterprise network, or secondary recursive servers operating in smaller sub-networks at the department level. I found that 301 of these 6,089 University hosts actively send queries (at least once every hour) over the day and contact more than 10 Internet-based DNS servers (resolvers or name-servers). These 301 hosts display the behavior of recursive resolvers but with fairly low throughput; thus, I deem them secondary resolvers. The remaining 5,788 hosts are only active for a limited interval (*i.e.,* between 5 min to 10 hours) and contact a small number of public resolvers (*e.g.,* 8.8.8.8 or 8.8.4.4 of Google) over the day.

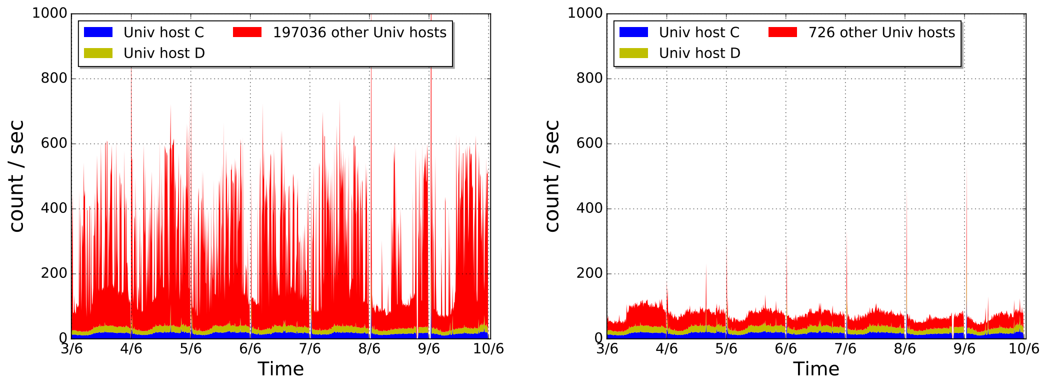
Response per host: Considering incoming responses (Fig. 3.2(b) for the university network), a larger number of "other" hosts in the organization are observed – approximately 196K IPs that are about full 3 "/16" subnets owned by the university. Most of these "other" hosts (*i.e.,* 97%) are the destinations of unsolicited responses, which indicates that either misconfiguration of external DNS servers, or the university network is suffering from DNS reflections.

Unwanted DNS packets per host: To better understand these potentially abnormal unanswered outgoing queries, unsolicited incoming responses, and error outgoing DNS lookups, I analyze their distribution among hosts in the two enterprises.

Fig. 3.3(a) shows the CCDF plot of the distributions per host for the university campus. All enterprise IP addresses in my dataset received unsolicited responses, and it is clear from the blue line that 99.9% of them are associated with 10 to 100 such packets – they did not have any outbound queries over the week. I observe that the hosts that have sent outbound queries to the public Internet received more unsolicited responses than those hosts that have never sent any DNS lookup. Outbound unanswered queries and error lookups are more concentrated on a small fraction of hosts, as shown in the tail of black and red lines. 2,140 and 1,812 (out of 6,091) hosts sent unanswered or error lookups – possibly due to packet drop during forwarding, typo-error of domain names, or malicious activities such as generating scans and DoS attacks.

Unsurprisingly, the primary recursive resolvers in both organizations are top sources and targets. In the University campus, hosts A and B respectively are the sources 4M (33%) and 3M (25%) unanswered queries, 12M (22%) and 10M (18%) error lookups, and are the destinations of 66K (3%) and 42K (2%) unsolicited responses.

Traffic composition of each host: Now I consider the distribution of normal outbound lookups, unanswered queries, error lookups, and unsolicited inbound responses within each host. Fig. 3.3(b) is the stack plot for the top 100 internal IP addresses with the most number of outgoing lookups (more than 35K over a week) with no error replies, and each bar represents an individual host. Seventy-three of them have more than 80% normal DNS packets in their outbound queries and inbound responses. The major unwanted DNS packet type is error lookups (red shades), such as `NameError`, `ServerFailure` and `QueryRefused`. It might be because of



(a) Incoming DNS queries. (b) Outgoing DNS responses.

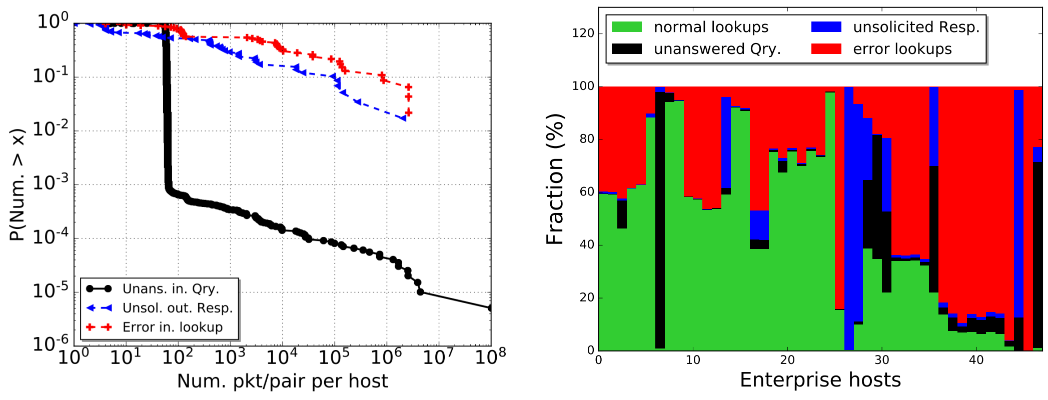
Figure 3.4: University campus: incoming queries and outgoing responses, measured during 3 June to 9 June 2019.

typo error in domain names or malicious DNS activities such as DoS attack or contacting remote attackers using random domain strings [179]. Unanswered queries (black shades) sent to external IP addresses that do not get a reply back are the second popular reason. Especially for hosts 13, 14, 15, and 19 with 23.5%, 23.2%, 23.3%, and 50.1% such unwanted outgoing queries, respectively – they are likely to be infected servers or host hackers that generating DNS scans or DoS attacks. Besides, three university hosts (order 56, 60, and 62) are also suffering from many unsolicited responses, occupying 9.29%, 3.96%, and 6.92% of their total number of packets for outbound queries and inbound responses. They suffered from small-scale DNS reflection attacks; for example, 99.04% unsolicited responses targeting host 56 are from only one recursive resolver configured by a private company located in China.

Incoming Queries & Outgoing Responses

Enterprises commonly receive DNS queries from the Internet that are addressed to their authoritative name servers.

It can be seen that two hosts of the University campus (*i.e.*, hosts C and D



(a) CCDF: # Unwanted DNS pkts per host. (b) Traffic composition per host.

Figure 3.5: University campus: (a) CCDF of # unwanted (incoming queries and outgoing responses) DNS packets and (b) their total fractions per host, measured during 3 June to 9 June 2019.

in Fig. 3.4(b)) are the dominant contributors to outgoing DNS responses – I have verified (by reverse lookup) that these hosts are indeed the name servers of the organization. Interestingly, for both organizations, I observe that a large number of hosts (*i.e.*, 197K IP addresses (shown by red shades in Fig. 3.4(a) for the university network) receive queries from the Internet. Still, a significant majority of them are unanswered (*i.e.*, 75.6%). These hosts are supposed to neither receive nor respond to incoming DNS queries, highlighting the amount of unwanted DNS traffic that targets enterprise hosts for scanning or DoS purposes.

Unwanted DNS packets per host: To better understand hosts involved in incoming queries and outgoing responses, I show the distribution of inbound unanswered queries, lookups replied with error responses and unsolicited outgoing responses from hosts inside the two enterprises.

Fig. 3.5(a) shows the CCDF plot of the distributions per host for the university campus. More than 99% enterprise IPs (including unassigned IP addresses) received unanswered queries from the Internet. As shown as the black line, almost all IPs are targeted by a small number (*i.e.*, less than 100) of such queries over a week – it indicates active and frequent DNS scans toward the organization. Some internal hosts received a massive amount of inbound queries at a high packet rate, located at the tail of the black line in Fig. 3.5(a), are likely to be victims of query flooding attacks. For example, a mixed DNS server (*i.e.,* performs as both authoritative name server and local recursive resolver) operated by a school in engineering faculty received 102M (75.5% of all unanswered incoming queries) lookups asking for non-enterprise services such as "`google.com`" and "`163.com`".

Moreover, 59 hosts sent unsolicited outbound responses (due to server misconfiguration, used as a reflector by internal attackers or packet drop); 47 hosts sent responses with errors (due to typos in domain names by outside users or being as victims in query-based attacks). In Fig. 3.5(a), the hosts that send unsolicited outbound responses are shown as blue dots, and the hosts that send responses with errors are shown as red dots. The top 3 hosts that sent most of the unsolicited responses (86.1%) are all servers operated by sub-department (verified by reverse lookups), and the organizational IT department does not have knowledge and control over them, highlighting the security blind spots for a large enterprise network.

Traffic composition of each host: Now I look at the distribution of normal inbound lookups, unanswered queries, outbound error lookups, and unsolicited responses within each host, as shown in Fig. 3.5(b) for 47 university hosts that sent outbound responses. It is clear that only six hosts are associated with more than 80% normal inbound lookup packets, and 45 hosts generated responses with error code other than `NoError`. Interestingly, 2,083 out of 2,085 outbound responses from the 45th host are labeled as error lookups – it is likely to be an idle authoritative name server, which received irrelevant questions such as "`researchscan541.eecs.umich.edu`", "`www.qq.com`" and "`www.wikipedia.org`" and respond with `QueryRefused`. Three hosts (ranked 26, 27, and 44 in terms of the number of outgoing responses) are occupied by more than 90% unsolicited responses. They are all operated by sub-departments and are potential error-configured (such as unsynchronized timing) or reflecting DNS responses for internal attackers, as I observed a significant amount of unsolicited responses for question name `miep` under the deprecated service type `ANY` and other irrelevant to the enterprise zone. Finally, three internal hosts suffered from a large fraction (more than 50%) of unanswered queries, especially for the 7th host – it is the mixed DNS server in engineering faculty as mentioned above, which was consistently under DoS attacks by irrelevant queries. The exhaustion of server resources led to it becoming unresponsive to most incoming queries (and only about 1% of queries got answered, including relevant and irrelevant questions).

3.4 Classifying Enterprise DNS Assets

In this section, I first articulate key attributes that can effectively differentiate types of DNS-related enterprise hosts (§3.4.1). I then develop a machine learning technique to determine if an enterprise host with a given DNS activity is a “name server”, “recursive resolver”, “mixed DNS server", or a “regular end-host” (§3.4.2). Finally, I rank the enterprise DNS servers into “name server” and “recursive resolver” by their importance, whereas mixed DNS servers are ranked in both types (§3.4.3).

My proposed system automatically generates lists of active servers into three categories located inside enterprise networks and rankings in terms of their name server and resolver functionalities, with the real-time DNS data mirrored from the border switch of enterprise networks. The system first performs ***“Data cleansing”*** that aggregates DNS data into one-day granularity and removes unsolicited responses and unanswered queries (*i.e.,* step 1); then ***“Attribute extraction”*** in step 2 computes attributes required by the following algorithms; ***“Server mapping”*** in step 3 classify DNS assets of various types; and finally ***“Server ranking”*** in step 4 ranks their criticality. The output is a classification and a ranked order of criticality, which an IT manager can then use to accordingly adjust management and security policies.

3.4.1 Attributes

Following the insights obtained from DNS behavior of various hosts, I now identify attributes that help automatically (a) map a given host to its function including authoritative name server, recursive resolver, mixed DNS server (*i.e.,* both name server and recursive resolver), or a regular client; and (b) rank the importance of DNS servers.

Dataset Cleansing

I first clean my dataset by removing unwanted (or malicious) records including unsolicited responses and unanswered queries – it removes the large fraction of unassigned or inactive IP addresses that are only associated with incoming DNS traffic. This is done by correlating the transaction ID of responses with the ID of their corresponding queries. In the cleaned dataset, incoming responses are equal in number to outgoing queries, and similarly for the number of incoming queries and outgoing responses.

Functionality Mapping

As discussed in §3.3, recursive resolvers are very active in terms of queries-out and responses-in, whereas name servers behave the opposite with high volume of queries-in and responses-out. Hence, a host attribute defined by the *query fraction of all outgoing DNS packets (**qryFracOut**)* should distinguish recursive resolvers from name servers. As shown in Table 3.4, this attribute has a value close to 1 for recursive resolvers and a value close to 0 for name servers.

In addition to recursive resolvers, there are some end-hosts configured to use public resolvers (*e.g.,* 8.8.8.8 of Google) that have a non-zero fraction of DNS queries out of the enterprise network. I note that these end-hosts ask a limited number of

Internet servers during their activity period whereas the recursive resolvers typically communicate with a larger number of external servers. Thus, I define a second attribute as the *fraction of total number of external servers queried (**fracExtSrv**) per individual enterprise host.* As shown in Table 3.4, the value of this attribute for end-hosts is much smaller than for recursive resolvers. Similarly for incoming queries, I consider a third attribute as the *fraction of total number of external hosts that initiate query in (**fracExtClient**) per individual enterprise host.* Indeed, this attribute has a larger value for name servers compared with other hosts, as shown in Table 3.4.

Lastly, to better distinguish between end-hosts and recursive resolvers (high and low profile servers), I define a fourth attribute as the *fraction of active hours for outgoing queries (**actvQryOutTime**).* For each host, this attribute indicate the fraction of time it sends outgoing queries. Regular clients have a smaller value of this attribute compared with recursive resolvers and mixed DNS servers, as shown in Table 3.4.

Importance Ranking

Three different attributes are used to rank the importance of name servers, recursive resolvers, and (non-DNS) public-facing servers respectively. Note that I rank mixed DNS servers within both name servers and recursive resolvers for their mixed DNS behaviour.

For recursive resolvers, I use ***QryFracHost*** defined as the *fraction of outgoing queries* sent by each host over the cleaned dataset. As for name servers, I use ***RespFracHost*** *as the fraction of outgoing responses* sent by each host.

Table 3.4: Samples of host attributes.

	qryFracOut	fracExtSrv	fracExtClient	actvQryOutTime
Univ name server (host C)	0	0	0.26	0
Rsch main name server	0	0	0.42	0
Univ recursive resolver (host A)	1	0.23	0	1
Rsch main recursive resolver	1	0.43	0	1
Univ mixed DNS Server	0.31	0.02	0.03	1
Rsch mixed DNS Server	0.23	0.0003	0.0013	1
Univ end-host	1	0.00001	0	0.041
Rsch end-host	1	0.00001	0	0.25

3.4.2 Host Clustering

I choose unsupervised clustering algorithms to perform the grouping and classification process because they are a better fit for datasets without ground truth labels but nevertheless exhibit a clear pattern for different groups/clusters.

Selecting Algorithms

I considered 3 common clustering algorithms, namely Hierarchical Clustering (HC), K-means and Expectation-maximization (EM). HC is more suitable for datasets with a large set of attributes and instances that have logical hierarchy (*e.g.,* genomic data). In my case however, hosts of enterprise networks do not have a logical hierarchy and the number of attributes are relatively small, therefore HC is not appropriate. K-means clustering algorithms are distance-based unsupervised machine learning techniques. By measuring the distance of attributes from each instance and their centroids, it groups data-points into a given number of clusters by iterations of moving centroids. In my case there is a significant distance variation of attributes for hosts within each cluster (*e.g.,* highly active name servers or recursive resolvers versus low active ones) which may lead to mis-clustering.

The EM algorithm is a suitable fit in my case since it uses the probability of an instance belonging to a cluster regardless of its absolute distance. It establishes initial centroids using a K-means algorithm, starts with an initial probability distribution

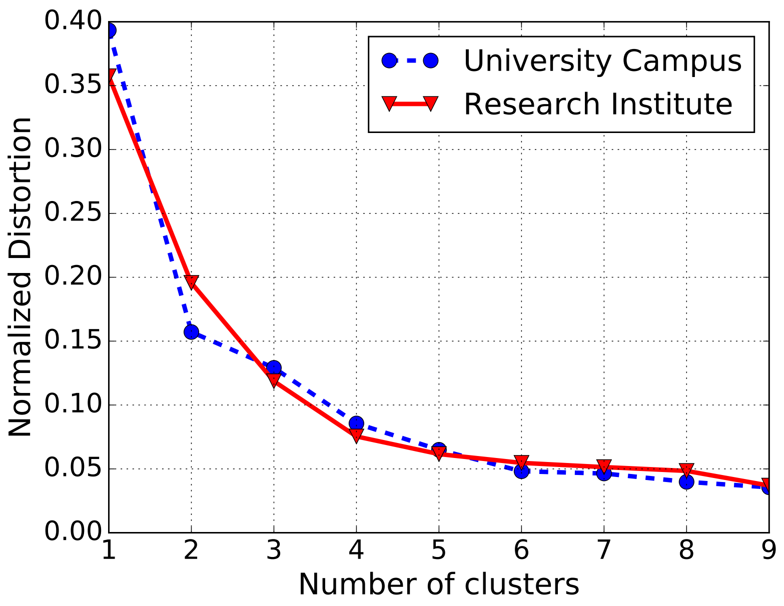


Figure 3.6: Elbow method: evaluating number of clusters.

following a Gaussian model and iterates to achieve convergence. This mechanism, without using absolute distance during iteration, decreases the chance of biased results due to extreme outliers. Hence, I choose an EM clustering algorithm for *"DNS Host Clustering Machine"*.

Number of Clusters

Choosing the appropriate number of clusters is the key step in clustering algorithms. As discussed earlier, I have chosen four clusters based on my observation of various types of servers. One way to validate the number of clusters is with the "elbow" method. The idea of the elbow method is to run k-means clustering on the dataset for a range of k values (say, k from 1 to 9 as shown in Fig. 3.6) that calculates the sum of squared errors (SSE) for each value of k. The error decreases as k increases; this is because as the number of clusters increases, the SSE becomes smaller so the distortion also gets smaller. The goal of the elbow method is to choose an optimal k around which the SSE reached its maximum absolute second order derivative, *i.e.*, the curve point that SSE decrease most rapidly. In Fig. 3.6, the curve points are

Table 3.5: University campus: host clusters (3 June 2019).

	Count	**qryFracOut**	**fracExtSrv**	**fracExtClient**	**actvQryOutTime**
name server	24	0.0004	1e-5	0.03	0.04
recursive resolver	21	0.99	0.04	6e-5	0.77
mixed DNS srv.	22	0.57	0.008	0.01	0.64
end-host	2,518	1.00	3e-5	0.00	0.24

Table 3.6: Research institute: host clusters (3 June 2019).

	Count	**qryFracOut**	**fracExtSrv**	**fracExtClient**	**actvQryOutTime**
name server	13	0.00	0.00	0.07	0.00
recursive resolver	25	1.00	0.03	0.00	0.86
mixed DNS srv.	2	0.81	0.05	0.04	0.54
end-host	245	1.00	5e-4	0.00	0.17

obtained at k=4 for both the university and the research institute, hence, four seems to be a reasonable number of clusters for both organizations.

Clustering Results

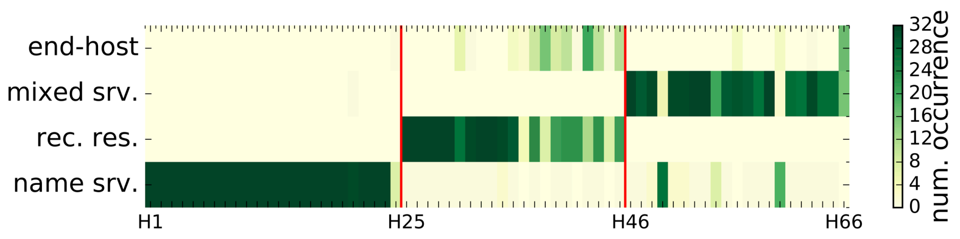
I tuned the number of iterations and type of covariance for my clustering machine to maximize the performance in both enterprises. Tables 3.5 and 3.6 show the number of hosts identified in each cluster based on data from 3 June 2019. I also see the average value of various attributes within each cluster. For the cluster of name servers, *qryFracOut* approaches 0 in both organizations (some name servers performed outbound DNS lookups for its own operational purposes), highlighting the fact that almost all outgoing DNS packets from these hosts are responses rather than queries, which matches with the expected behavior. Having a high number of external clients served also indicates the activity of these hosts – in the University campus and research institute respectively 24 and 13 name servers collectively serve 81.6% and 91% (*i.e.,* 24×3.4% and 13×7%) of external hosts.

Considering recursive resolvers in Tables 3.5 and 3.6, the average *QryFracOut* is close to 1 for both organizations as expected. It is seen that some of these hosts also answer incoming queries (from external hosts) possibly due to their misconfiguration. However, the number of external clients served by these hosts is very small (*i.e.,* less than 5 per recursive resolver) leading to an average fraction near 0. Also, looking at the number of external servers queried (*i.e., fracExtSrv*), the average value of this attribute for recursive resolvers is reasonably high, *i.e.,* 21 and 25 hosts in the University and the research network respectively contribute to 83% and 89% of total *fracExtSrv* – this is also expected since they commonly communicate with public resolvers or authoritative name servers on the Internet.

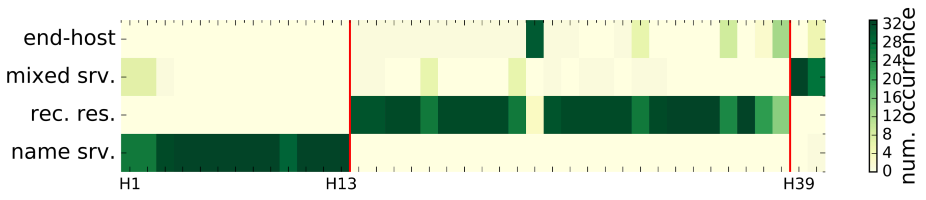
Hosts clustered as mixed DNS servers in both organizations have a moderate value of the *QryFracOut* attribute (*i.e.,* 0.57 and 0.81 for the University and the research network respectively) depending on their varying level of inbound/outbound DNS activity. Also, in terms of external clients and servers communicated with, the mixed servers lie between name servers and recursive resolvers. Lastly, regular end-hosts generate only outbound DNS queries (*i.e., QryFracOut* equals to 1), contact a small number of external resolvers, and are active for shorter duration of time over a day (*i.e., actvQryOutTime* less than 0.5).

Interpreting the Output of Clustering

My clustering algorithm also generates a confidence level as an output. This can be used as a measure of reliability for my classifier. If adequate information is not provided by attributes of an instance then the algorithm will decide its cluster with a low confidence level. The average confidence level of the result clustering is 98.13% for both organizations, with more than 99% of instances classified with a confidence-level of more than 85%. This indicates the strength of my host-level attributes, enabling the algorithm to cluster them with a very high confidence-level.



(a) Univesity campus.



(b) Research institute.

Figure 3.7: Hosts clustering results across 32 days.

Server Clusters Across 32 Days

I now check the performance of my clustering algorithm over 32 days. Fig. 3.7 shows a heat map for clusters of servers. Columns list server hosts that were identified in Tables 3.5 and 3.6 (*i.e.,* 66 hosts in the University network and 40 hosts in the research network). Rows display the cluster into which each server is classified. The color of each cell depicts the number of days (over 32 days) that each host is identified as the corresponding cluster – dark cells depict a high number of occurrences (approaching 32), while bright cells represent a low occurrence closer to 0.

In the University network I identified 25 name servers, shown by H1 to H25 in Fig. 3.7(a); the majority of which are repeatedly classified as a name server over 32 days, thus represented by dark cells at their intersections with the bottom row, highlighting the strong signature of their profile as a name server. An exception is H25, which was only active for 7 days as name server and 1 day as end-host. It is an IP addresses belonging to school of physics under department of science, as verified by reverse lookups.

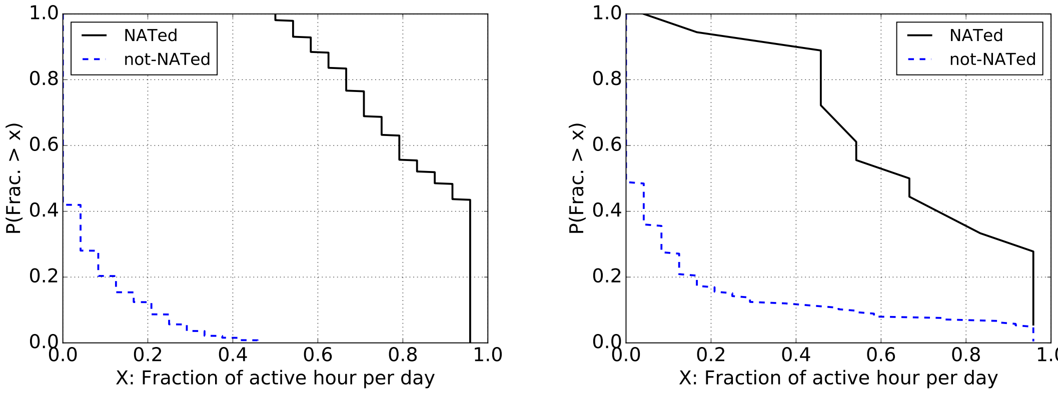
Among 21 recursive resolvers of the university campus, shown by H25 to H46 in

Fig. 3.7(a); 7 of them (including hosts A and B in Fig. 3.2) are consistently classified as recursive resolver, and the rest are re-classified as end hosts (due to their varying activity). Lastly, 20 mixed servers, shown by H46 to H66 in Fig. 3.7(a), are classified consistently though their behavior sometimes is closer to a end-host or a name server.

My results from the Research Institute network are fairly similar – Fig. 3.7(b) shows that hosts H1-H13 are consistently classified as name servers, while hosts H14-H38 are recursive resolvers and H39-H40 are mixes servers. Unlike the University Campus, 9 recursive resolvers are classified as mixed-server from 1 to 6 days. They are owned by business units in the organization, revealing the dynamicity of their DNS infrastructures.

Clustering of End-hosts: NATed or Not?

Knowing whether an end-host is a standalone device or a NATed gateway that represents multiple users is also important for enterprise network management. For example, an IT department could block the Internet connectivity of a standalone device that has anomalous activities, while such strict measure may not be suitable for a NATed device since it also serves many benign hosts. To draw more insights I further applied my clustering algorithm (using the same attributes introduced in §3.4.1) to IP addresses of end-hosts, determining whether they are behind a NAT gateway or not (*i.e.,* two clusters: NATed and not-NATed). In both networks, all WiFi clients are behind NAT gateways. Additionally, some specific departments of the two enterprises use NAT for their wired clients too. I verified my end-host clustering by reverse lookup in the respective enterprise network. Each NATed IP address has a corresponding domain name in specific forms as configured by IT departments. For example, the University campus wireless NAT gateways are with their domain-names as "`SSID-pat-pool-a-b-c-d.gw.unsw.edu.au`", where "`a.b.c.d`" is the public IP address of the NAT gateway, and "`SSID` is the the WiFi SSID for the University campus network. Similarly, in the Research institute, NAT gateways



(a) University campus. (b) Research institute.

Figure 3.8: CCDF: fraction of active hour per day for NATed and not-NATed end-host IP addresses.

use names in the form of "`c-d.pool.rsch-primary-domain`" where "`c.d`" is the last two octets of their public IP addresses.

On 3rd June 2019, my end-host clustering shows that 337 and 42 of end-hosts IP addresses are NATed in the University campus and the Research institute, respectively. I note that the two clusters of end-hosts are distinguished primarily by two attributes, namely *actvTimeFrac* – a NATed IP address (representing a group of end-hosts) is expected to have a longer duration of DNS activity compared to a not-NATed IP address (representing a single end-host), as illustrated in Fig. 3.8; and *numExtSrv* – a NATed IP address is expected to have more than one queried public DNS resolvers, as it represents many individual hosts each connected with their selected resolvers on the Internet. All classified not-NATed hosts contacted less than 10 external DNS servers in both organizations during 3rd June, while 54% and 26% NATed IPs in the university and research institute queried more than 10 public servers.

I verified their corresponding domain names configured by their IT departments. Some IPs with domain-names of NAT gateways are incorrectly classified as not-NATed end-hosts. This is because their daily DNS activity was fairly low, *i.e.*, less than an hour with only one external resolver contacted. On the other hand, not-

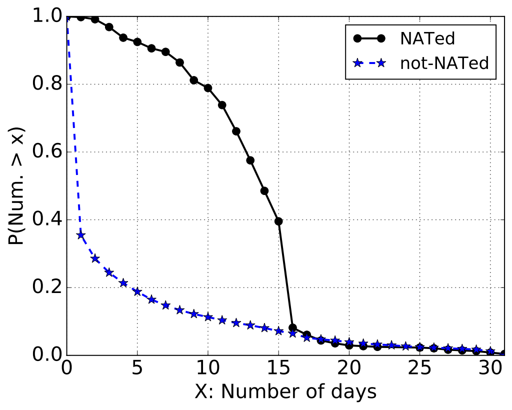
NATed end-hosts with long duration of DNS activity (*i.e.,* almost the whole day) were misclassified. Verifying end-hosts classified as NATed, 77.2% of them in the University campus and 75.0% in the Research institute have corresponding domain-names as for NAT gateways allocated by IT departments. For end-hosts classified as not-NATed, 91.1% and 93.6% in the respective two organizations do not map to any organizational domain-names.

Looking into the consistency of end-hosts clustering across 32 days, I note that more than 90% end-hosts in the University campus are consistently labeled as NATed over 7 days (as show in Fig. 3.9(a)). 52% end-hosts are classified as NATed from 7 days to 15 days. Those IP addresses are owned by sub-departments in the university, and re-shuffled within their subnets by the organizational DHCP servers periodically. As for the University IP addresses get classified as not-NATed (*e.g.,* desktops with public IP addresses through wired connection), majority (63%) of them only appear once during 32 days. It is because of their low-profile activities and daily IP re-shuffling.

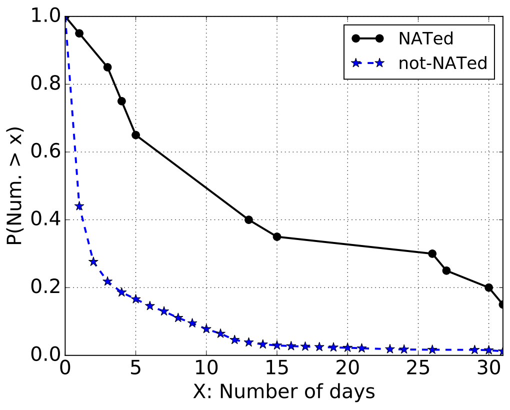
Similar observations were obtained from the research institute (shown in Fig. 3.9(b)), except that there are five IP addresses appeared as NATed across the 32 days – they belong to IT infrastructures controlled by critical scientific basements such as Australia Telescope National Facilities, which are separated controlled with more freedom thus not affected by periodically DHCP reallocation.

IT Verification

The IT department in both organizations verified the top-ranked DNS resolvers (two in the University and one in the Research Institute) and name-servers (two in the University and one in the Research Institute) found across the 32 days, meaning 100% accuracy for ground-truth DNS assets, as they are directly configured and controlled by the IT departments. Additionally, as will be discussed next in



(a) University campus.



(b) Research institute.

Figure 3.9: CCDF: Consistency of end-hosts clustering across 32 days.

§3.4.3, my method revealed unknown name servers, recursive resolvers, and mixed DNS servers configured by departments of the two enterprises (I verified their functionality by reverse DNS lookup and their IP range allocated by IT departments). Interestingly, 3 of the name-servers my method identified were involved as reflectors in a DNS amplification attack, and IT was able to confirm that these were managed by affiliated entities (such as retail stores that lease space and Internet connectivity from the University) - this clearly points to the use of my system in identifying and classifying assets whose security posture the network operators themselves may not have direct control over.

3.4.3 Server Ranking

My system discovered 46 authoritative name servers and 43 recursive resolvers in the University (a mixed DNS server are treated as both name server and recursive resolver), and 15 authoritative name server and 27 recursive resolvers at the Research Institute. However, only 6 top ranked DNS servers, in each organization, contribute to more than 90% of outgoing queries and responses. Servers ranking provides network operators with the popularity of their DNS assets.

3.5 Monitoring DNS Asset Health

Having shown how DNS assets in an enterprise network can be identified and classified based on their network behavior, I now extend the study to monitor their health on a continuous basis. The objective is to detect *anomalous* behavior, indicating that the asset is being misused or attacked, and identify the root cause of such deviations in behavior. I begin in §3.5.1 by providing two examples of observable anomalies from my dataset – one attributable to poor configuration, and the other subject to a DDoS attack. Inspired by these examples, in §3.5.2 I suggest a set of *health metrics* that can track the behavior of each asset along various dimensions, and in §3.5.3 develop a method to label and alert anomalous behaviors based on these continuous health tracking metrics. Finally, in §3.5.4 I apply my methods to the 32-day dataset from the two organizations and present results into misuse and attack patterns detected by my methods.

3.5.1 Examples Illustrating Anomalous DNS Asset Behavior

By manually inspecting my dataset, I could identify several behavioral patterns that seemed unusual. Therefore, I begin by providing a couple of illustrative examples of anomalous behavior and subsequently develop methods to automatically detect misbehaviors by tracking various health metrics.

Example 1 – DNS misuse: I found that one of the Authoritative DNS Severs from the Research Institute dataset was responding with an unusually high number of "`NXDOMAIN`" messages. Upon investigation, I found that it was being queried with names that were irrelevant to the organization, *e.g.,* "`www.taobao.com`". In fact 31.8% of DNS queries to this server were irrelevant to the organization, yet it was responding; a further 14.2% were malformed (*e.g.,* "`com`"), to which the server was responding with the "`NXDOMAIN`" message. This obviously represents a poorly configured server that is readily responding to every irrelevant and malformed query, rather than just the domains it is authoritative for – this exposes the server for attckers to launch a denial-of-service attack, or to use it as a reflector for attacking others.

Example 2 – DNS flood attack: I found one of the Authoritative DNS Servers in the University dataset to show a sustained 142% increase in inbound query rates over a 10-day period (7-Jun 0:17AM till 17-Jun 4:43PM). Upon closer inspection, I noted 3.3M queries, all with the same query name "`aids.gov`", had come during this period from 974 external sources – typically each external source launched around 300 queries within a 20 second period, and then went idle. The DNS server was unable to keep up with the high rate of requests, and was able to service only about 70% of incoming queries. Further, 40.9% of the responses during this period were irrelevant to the organization, while 21.6% were with the "`NXDOMAIN`" error.

3.5.2 DNS Traffic Health Metrics

Having seen some examples of poor behavior from DNS servers, I now propose several metrics that can be used to track the health of each DNS asset in the organization. I categorize them into service, functional, network, and volumetric behaviors.

Service behavior: From a border perspective, authoritative name servers are expected to only serve DNS queries seeking to resolve domains relevant to the enterprise. Conversely, recursive resolvers should only send outbound queries for domains outside of the enterprise – queries for internal domains are internally sent to the enterprise authoritative name servers without crossing the network border. We therefore define `Non-Enterprise Lookup Fraction (NELF)` as the fraction of query names that are irrelevant to the enterprise services. A properly configured authoritative name server should have `NELF` of 0, while for a recursive resolver this metric should be 1.

Functional behavior: Under ideal conditions, responses of a properly functioning DNS server are expected to carry "`NoError`" as response code. However, a DNS query may fail due to some errors, such as the domain name queried does not exist, an answer cannot be given, or the server refuses to answer due to policy. Therefore, we define `Lookup Error Fraction (LEF)` for a DNS server as the fraction of its responses that carry an error code – a value significantly larger than 0 indicates potential misbehavior.

Network behavior: Under normal circumstances a query is associated with a response. However, the network trace often reveals inbound responses with no outbound queries (*e.g.,* a reflective attack to a victim whose IP address was spoofed), as well as outbound queries with no inbound response (*e.g.,* a malicious internal host launching a DoS attack via the DNS cache/proxy). To track such anomalous network behavior, we define the Query Service Ratio Inbound (QSRI), *i.e.,* ratio of outbound responses to inbound queries, and Query Service Ratio Outbound (QSRO), *i.e.,* ratio of inbound responses and outbound queries. All DNS assets should ideally have these two metrics as 1, showing the balanced profile of queries and responses.

Volumetric behavior: Sudden increases in the rates of DNS packets can indicate that enterprise assets are being targeted by attacks. I therefore track the inbound and outbound rates of queries and responses for each DNS asset over each epoch (of one hour), respectively `QryRateIn (QRI)`, `RespRateOut (RRO)`, `QryRateOut (QRO)`, and `RespRateIn (RRI)`, and flag those epochs in which the rate shows an increase above a prescribed threshold value (discussed below), which could be indicative of volumetric attacks.

3.5.3 Using Health Metrics to Detect Anomalies

Using the health metrics identified above, I build a simple mechanism to detect and alert various anomalous behaviors of DNS assets. The set of anomalies I consider in

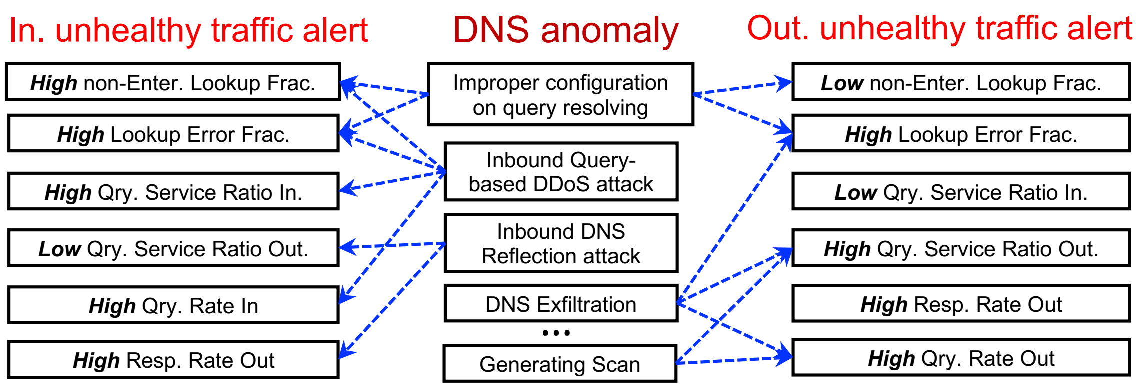

In. unhealthy traffic alert
DNS anomaly
Out. unhealthy traffic alert
High non-Enter. Lookup Frac.
High Lookup Error Frac.
High Qry. Service Ratio In.
Low Qry. Service Ratio Out.
High Qry. Rate In
High Resp. Rate Out
Improper configuration on query resolving
Inbound Query-based DDoS attack
Inbound DNS Reflection attack
DNS Exfiltration
...
Generating Scan
Low non-Enter. Lookup Frac.
High Lookup Error Frac.
Low Qry. Service Ratio In.
High Qry. Service Ratio Out.
High Resp. Rate Out
High Qry. Rate Out

Figure 3.10: Some examples of my observed DNS anomalies and their related traffic health alerts.

this chapter, illustrated pictorially in Fig. 3.10, include:

- **Misconfiguration:** Consider an authoritative DNS server that has been poorly configured and resolves queries for domains that it has no authority over (*i.e.*, do not belong to the enterprise). The exploitation of this by attackers (*e.g.*, as a reflector) will manifest in an alert when the NELF metric becomes high, while LEF could also be high (in case the queries are malformed or non-existent). Conversely, a misconfiguration alert is triggered when the NELF metric falls below a threshold value for a poorly configured recursive DNS resolver.

- **DDoS Attack:** A distributed denial-of-service attack on an enterprise DNS server will manifest in the form of a volumetric rise in QRI, potentially accompanied by a high value in NELF and/or LEF. Most queries in DDoS attacks tend to be either fixed or random domains instead of customizing query names specific to the victim enterprise.

- **Reflection attack:** An inbound reflection attack on an enterprise asset usually targets the DNS cache/proxy, by bombarding it with unsolicited traffic. This will manifest in the form of a rising incoming response rate (RRI), as well as low ratio of outbound queries to incoming responses (QSRO).

- **DNS exfiltration:** An infected enterprise host that is trying to exfiltrate data via DNS will cause the QRO to rise, potentially accompanied with unanswered queries (rise in QSRO) and/or lookup errors (rise in LEF). These can be used as triggers to conduct deeper investigation into exfiltration, *e.g.,* using my method in [201]. One may argue that QRO is expected to be relatively high for legitimate recursive resolvers. Therefore, we infer from a combination of metrics, each with specific thresholds (value ranges) to cater for some reasonable deviations (discussed in §3.5.4).

- **Scans:** Presence of malware in the enterprise that performs outbound scans can be detected by monitoring for a rise in outbound queries (QRO), potentially accompanied with unanswered queries (rise in QSRO) and/or lookup errors (rise in LEF).

In what follows I continuously track the health metrics of the various DNS assets identified in the two enterprise networks by my earlier clustering algorithm, and evaluate my ability to identify anomalous behaviors indicative of misconfigurations and/or attacks. Note that our proposed metrics and alerts from DNS behavioral monitoring could be consumed by SIEM platforms and/or combined with security appliances to verify whether an enterprise host is indeed involved in malicious communications or not. Such combined inferences are beyond the scope of this thesis.

3.5.4 Insights in two Enterprise Networks

I applied the proposed traffic health metrics to my 32-day DNS traces captured from both organizations, comprising the assets as identified earlier in Tables 3.5 and 3.6 for the University (67 DNS assets) and Research Institute (40 DNS assets) respectively. The metrics are computed each epoch (of one hour), and my first step is to identify epochs wherein the health metrics deviate significantly from their expected values. In general, DNS assets in the University raise more alerts than

Table 3.7: Alerts and occurrence frequency (in the fraction of epochs) for my two example DNS assets.

Direction	**Profile**	**Alert**	**Example 1**	**Example 2**
Inbound ↓	Service	high NELF	83.7%	85.6%
↓	Functional	high LEF	6.6%	0.1%
↓	Network	low QSRI	94.5%	15.6%
↓	Network	high QSRO	0.1%	0.3%
↓	Volmetric	high QRI	0.0%	29.7%
↓	Volmetric	high RRI	5.1%	9.8%
Outbound ↑	Service	low NELF	0.0%	0.0%
↑	Functional	high LEF	100.0%	20.2%
↑	Network	high QSRI	0.8%	0.3%
↑	Network	low QSRO	0.0%	84.4%
↑	Volumetric	high RRO	5.6%	7.7%
↑	Volumetric	high QRO	1.0%	31.1%

the research institute. In order to limit the number of alerts, I choose a margin value that is at the elbow points in a curve, which is at around the 30% mark. This is also consistent with the threshold values used by many state-of-the-arts security appliances, *e.g.,* from Palo Alto [202], Fortinet [203] and Cisco [204]. While organizations are free to tune the threshold alerting values for each health metric to suit their environments, in this work for simplicity I will maintain it at 30%. In what follows I first examine two DNS assets that exhibited high rates of alerts (as shown in Table 3.7), followed by a general overview of alerts across the two organizations. I then design an inference engine that combines the health metric alerts and deduces the nature of the underlying anomaly causing these alerts using the relationships identified earlier in §3.5.3.

Example 1: A DNS server in the University Law Department serves as both authoritative name server and recursive resolver. It exhibited unhealthy elevated NELF metric for 83.7% of epochs, and unhealthy depressed QSRI for 94.5% of epochs, indicating its **misconfiguration** was being exploited by attackers for a potential **DDoS attack**. Queries for "`d.c.b.a.in-addr.arpa`" were coming from many

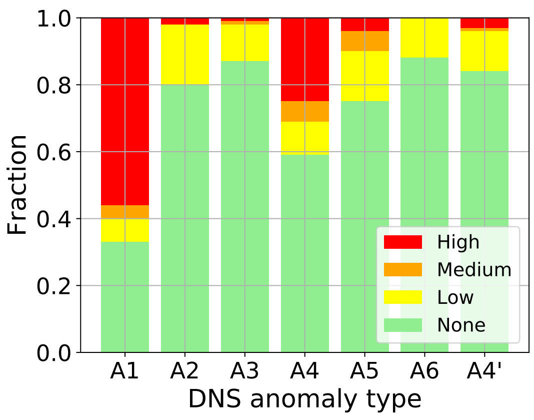
Table 3.8: DNS anomalies considered in this chapter and their indicative alerts and required post-hoc analysis.

DNS Anomaly Type	Indicative Alerts	Post-hoc Analysis
A1: Misconfiguration	↑ NELF & LEF	None
A2: Query DDoS	↑ QSRI & QRI	Flow profile
A3:Response DDoS	↓ QSRO & ↑ RRI	Flow profile
A4: Attack reflector	↑ QRI & RRO	Flow profile
A5: Generating scan	↑ LEF & ↓ QSRO	Flow profile
A6: Data exfiltration	↑ LEF & QRO & ↓ QSRO	Query content
A4': Reflector (after fix)	↑ QRI & RRO	Flow profile

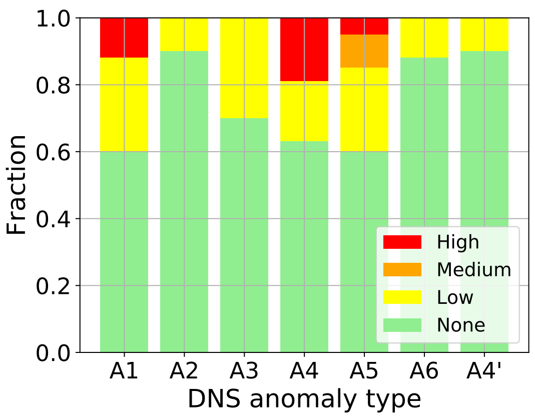
external IP addresses, and the server was responding to a vast majority (over 90%) of them, thereby wasting its resources. The asset also exhibited many epochs (6.6%) of unhealthy LEF metric, indicating that it might proxy **scans**. On 29-Jun, this server sent queries to 131 external IP addresses, of which 18 responded – this asset is likely being utilized as a proxy to perform slow reconnaissance scans to discover availability of DNS servers on the Internet.

Example 2: A DNS server in the University Engineering Department also exhibited many inbound health alerts, such as high NELP for 85.6% of epochs, low QSRI for 15.6% of epochs, and high QRI for 29.7% of epochs. Investigation confirmed that it was **misconfigured** and exploited by attackers using it to launch reflection attacks with queries for domain names such as `dnsscan.shadowserver.org`, `researchscan541.eecs.umich.edu`, and `nil`. The sever was also giving outbound health alerts for high LEF, QSRO, and QRO, indicating a potential for **DNS exfiltration**. Indeed, my post-hoc analysis showed that on 30-Jun it sent out 709K DNS queries with pattern `SARICA[10digits].com` towards an IP address in Turkey, and on the next day, another 964K DNS queries to the same server with pattern `akbank[9digits].com.tr` – the random 9 or 10 digits very likely encode exfiltrated data as highlighted in [201].

Alerts across the two organizations: Certain DNS assets – 35% in the University and 13% in the research institute – were consistently flagged by alerts in each epoch. These turn out to be largely Authoritative DNS servers that are



(a) Univesity campus.



(b) Research institute.

Figure 3.11: Severity of DNS anomalies of each enterprise asset in both organizations.

publicly facing, and hence exposed to inbound DNS attacks (interesting, most of these were managed by sub-departments or third-parties, rather than central IT in the organization). Recursive resolvers in both organizations raised relatively fewer alerts, typically in `QRO` and `RRI` during some epochs.

Inferring anomalies from alerts: Tracking the health metrics (aka "symptoms") allows us to make inferences about the underlying anomalies (aka "diseases"). I built a simple inference engine using the Codebook Correlation technique used extensively in Network Management for event correlation [205]. A causality graph was built as per Fig. 3.10, a codebook correlation model was derived, and then "alerts" from the 32-day dataset were looked up in the codebook to determine the underlying "anomaly". The outcomes, in terms of the health of the DNS assets across the two organizations, are shown in Fig 3.11.

My first observation is that misconfiguration is a significant problem across both organizations – 56% and 33% of DNS assets in the University and research institute, respectively, serve DNS queries not relevant to the enterprise. This is a serious concern – Authoritative DNS servers are resolving non-enterprise queries and thereby being exposed to random queries, which can lead to denial-of-service; while recursive resolvers are resolving queries for non-enterprise hosts, thus being made available to attackers as reflectors for DDoS attacks on spoofed victims. Indeed, my analysis shows that if these DNS configurations were to be rectified, the number of DNS assets being used as reflectors falls from 25% to 3% in the University, and from 20% to 0% in the Research Institute (as shown in the rightmost bar of Fig 3.11).

The second most significant concern is that there is evidence of scans emanating from both organizations, as indicated by epochs of high lookup failures (LEF) and low success of responses (QSRO). These indicate that there is malware lurking within the organizations that is using DNS to perform scans on other Internet hosts. Identifying malware-infected hosts would require access to traffic within the organization, which is beyond the scope of this chapter.

Finally, I note that there are epochs with evidence of DNS data exfiltration from the University network. Again, knowing the hosts complicit in this requires analysis of traffic within the organization (my traffic feed at the border does not tell us which internal host made the DNS request to the organizational cache/proxy), which is beyond the scope of this chapter. Similarly, a few assets in the Research Institute are occasionally launching DDoS attacks on external victims.

While I do not intend to diagnose every DNS problem, it is continuously assessing the health of each DNS asset in the organization, and flagging potential issues that can be investigated further by the network operator, providing them actionable intelligence to rectify misconfigurations, amend firewall policy rules, rate-limit query rates, etc., to better protect their assets.

3.6 Conclusion

Enterprise networks are often vulnerable to DNS-based cyber attacks due to insufficient monitoring of DNS traffic. In this chapter, I have developed methods to classify enterprise assets and continuously track their cyber-health by passively analyzing DNS traffic crossing the network border of an organization. I performed a comprehensive analysis of DNS packets from two large organizations to identify asset profiles by network, functional, and service characteristics. I highlighted the behavior of enterprise hosts, either benign and anomalous. I then trained unsupervised machine learning models by DNS traffic attributes that classify the DNS assets, including authoritative name server, recursive resolver, mixed DNS server, and end-hosts behind or not behind the NAT. Lastly, I developed metrics to track the cyber health of enterprise DNS assets continuously. I identified several instances of improper configurations, data exfiltration, DDoS, and reflection attacks. Results of my real-time application have been verified with IT departments of the two organizations while revealing unknown knowledge that helps them enhance their security management without incurring risks and excessive labor costs.

Chapter 4

Classifying and Tracking Enterprise Assets via Network Behavioral Analysis

Contents

Enterprise networks continue to grow in scale and complexity, encompassing a wide range of connected end-points including web servers/proxies, DNS servers/proxies, VPN/mail servers, and other special-purpose devices. Monitoring this dynamically evolving set of assets, for the purposes of ensuring operational efficiency as well as cyber security, poses a significant challenge for IT personnel. In this chapter I develop, prototype, and evaluate a system that automatically identifies and classifies enterprise connected assets in a continuous manner by analyzing their network activity, thereby reducing blind spots for organizational IT departments.

My contributions are three-fold: (1) I conduct off-line analysis on traffic traces of over 3 billion packets taken from a large enterprise network to deduce fine-grained behavioral profiles of the most popular asset types like website servers, DNS servers, and file storage systems and transport-layer communication patterns of less popular ones such as non-standard TCP/UDP servers, proxies, and NAT gateways; (2) I systematically develop host-level behavioral attributes, train multi-class classifiers in a multi-grained classification scheme to categorize connected assets, and evaluate them via cross-fold validation as well as open set to demonstrate overall accuracy of more than 98%; and (3) I prototype my system using software-defined networks, deploy it to operate for a month on multiple 10Gbps Internet links of a real enterprise network, and present insights such as the ability to identify hundreds of typical servers and their utilization, as well as thousands of non-typical assets, and highlight anomalous behaviors pertinent to possible cyber-threats. My solution provides a dynamic and scalable way for IT personnel to reduce their blind spots in effectively tracking enterprise assets.

4.1 Introduction

Enterprise networks host a variety of connected devices ranging from website servers, web proxies, DNS proxies, mail servers, and VPN servers to remote computing platforms and desktops. Organizational IT departments struggle to keep track of their complex environment [163, 206], which continuously evolves as assets get decommissioned, and new ones are added. Consequently, it is common for organizations to be unaware of under-utilized and orphaned assets contributing to operational inefficiencies, as well as diverse device-specific vulnerabilities exposing the organization to cyber threats. First-hand experience has shown that the problem is particularly acute in Universities, wherein connected assets are managed in a loosely federated manner across departments. For instance, the requirements of research groups limit the ability to enforce a standard operating environment (SOE), and the culture is attuned to staff and students connecting their own devices into the campus network. These blind spots have handicapped IT departments and exposed Universities to cyber-risks ranging from malware and botnets to reflection-based DDoS attacks [207].

Maintaining an up-to-date inventory of connected assets is very challenging in practice. Manually updated spreadsheets become obsolete very quickly and are rarely synchronized across IT team members, let alone across the organization (for example, Facilities Management in my University deploys security cameras and smart monitors, and IT only finds out later). Enterprise IT managers have built home-grown tools to track assets by ingesting logs from various network services like DHCP servers, RADIUS authentication servers, DNS servers, firewalls, and web proxies; however, each of these provides a narrow and independent view of the connected assets [208, 209], leaving many hosts undiscovered. This can be very problematic, given that enterprise operators largely rely on device-specific rules (*e.g.,* access control lists) on their border security appliance (*e.g.,* firewall) to protect their assets, and having blind spots in their assets inventory can result in exposing a large attack surface [163, 210, 211].

Prior research works on classifying connected hosts and monitoring their behavioral patterns have relied on either: (a) processing packet contents for "signatures" [97, 212, 213]; or (b) deducing end-to-end communication graph patterns to identify host types by correlations of networked entities [64, 70, 75, 76, 214–217]. With the increasing adoption of packet encryption, packet content inspection is becoming infeasible. With enterprise assets numbering tens of thousands and traffic rates growing to tens of Gbps, construing a reliable graph structure of all communication flows between internal hosts, and external services can be computationally very expensive.

Therefore, in this chapter, I propose a new method that leverages programmable networks capability and data-driven learning algorithms to analyze network traffic in real-time. Specifically, my method: (1) captures transport-layer behavior for all enterprise hosts without any packet content inspection, and analyzes these to classify with high confidence a vast majority of assets that exhibit well-known behavioral patterns; and (2) conducts a deeper investigation of only a small dynamic subset of hosts exhibiting unfamiliar behavior to identify their non-standard roles (such as a web server using non-typical ports). My approach has the advantage of being encryption-resistant, scalable, and easy to deploy without putting network operations at risk. My work makes three specific contributions elaborated below:

First, by analyzing a large representative traffic trace of over 3 billion packets collected from a University campus network, I summarize the typical network behavior of the ten most common host types in an enterprise network, such as web server, mail server, and DNS server. I identify the presence of non-typical hosts such as servers that use less common protocols or non-standard services (ports), and hosts that serve multiple purposes. I further categorize enterprise hosts into six coarse-grained types based on their distinct transport-layer behaviors, including: TCP- and UDP-based public-facing servers that provide enterprise content to external users; TCP- and UDP-proxies that act as relays (*e.g.,* for DNS and HTTPS); NAT gateways that represent internal clients with private IP addresses; and end-hosts that have unique public-facing IP addresses.

Second, I develop a multi-grained classification scheme that uses a rich set of host-level attributes and supervised machine learning (ML) models to deduce the role of enterprise hosts. To this end, I first systematically profile host network behavior using a host-specific rooted-graph structure and identify descriptive attributes of network behaviors. I then optimize my data structures and behavioral attributes by balancing their predictive power against computational cost. I lastly develop ML models by tuning various algorithms, model parameters, input attribute sets, and retention periods. My fine-grained model classifies enterprise hosts into N common specific types ($N = 10$ in my use-case), and the coarse-grained model classifies hosts into six generic types labeled by their dominant services, highlighting their functionality. Well-tuned models yield a high accuracy (close to 99%) in cross-fold validation while providing cost- effectiveness (scalable and practical).

Third, I prototype my system using a commodity programmable hardware switch and virtual network functions (VNF) on a generic server. I deploy it at the edge of my University campus network, and highlight insights obtained over a one-month trial period. My system was able to uncover over 300 web servers, along with several DNS servers, mail servers, NAT gateways, and proxies. Additionally, my system was able to detect unexpected behavior from several assets indicative of scans and malware, though a deeper study of these is beyond the scope of this thesis. I also profile the performance of my solution in terms of CPU and memory usage, responsiveness, and inspection load, validating that the system can easily scale to large enterprises.

The rest of this chapter is organized as follows: §4.3 describes insights from my analysis of a traffic traces of over 3 billion packets captured from the Internet border router of my university campus network. In §4.4 I describe attribute extraction, machine learning model training, and my multi-grained scheme for enterprise host classification based on network behavior. Prototype design, development, and evaluation is described in §4.5. Prior work is summarized in §4.2, and the chapter is concluded in §4.6.

4.2 Related Work

I now discuss prior works on topics, including analysis of network traffic, classification of host behaviors, and systems with programmable networks, highlighting the novelty of my work.

4.2.1 Analysis of Network Traffic

Analysis of network traffic has been a hot topic for more than two decades. Port-based analysis methods [218, 219] that map traffic type by source/destination transport-layer port numbers are widely used in many commercial solutions, for its low computational cost and high accuracy in identifying certain application types and servers like website (HTTP/HTTPS) and name resolutions (DNS). However, with the complexity of modern applications and host roles in using a variety of transport services [220], purely port-based approaches fall short in effectively classify all hosts in an enterprise network. Analysis of network traffic using statistical methods such as machine learning is increasingly gaining interest from the research community [221–223]. Several works have been done in detecting and classifying various types of traffic such as video streaming [224], coflows from cloud computing [225], DNS [2], and email service [226].

4.2.2 Classification of Host Behaviors

There are many research works on classifying host types [64, 70, 75, 76, 214, 217, 227]. Work in [70] constructed networked graphs that represent interconnections between hosts; *G. Tan et al.* [215] proposed two effective algorithms in obtaining the similarity between host communication patterns to identify their social groups; and *BLINC* [64] classified host types using flow statistics to summarize host roles including attackers and victims in cyber-crimes. Authors of [76] utilized a stochastic block model to identify pattern changes in the networked graph of host connections for potential anomalous changes. *Baywatch* [228] identified malware-infected hosts in an enterprise network by analyzing their beaconing behavior (the process for infected hosts communicating with remote Command-and-Control servers). *Beehive* [216] performed large-scale analysis on logs collected from key IT infrastructures such as DHCP servers, VPN gateways, and web proxies to detect suspicious host activities.

4.2.3 Systems with Programmable Networking

Software-define and programmable networking techniques (*i.e.,* SDN and NFV) have been employed to address various research problems such as dynamic telemetry and security enforcement [125]. Their use-cases range from measuring network-wide flow-level statistics elastically [229, 230], satisfying operators' dynamic needs for network telemetry via general-purpose query-driven system [86], identifying and fingerprinting specific network traffic like video streaming [224, 231] to detecting network attacks and threats reactively [11, 140, 232].

4.2.4 My Key Novelty

My first key novelty stems from profiling the network behavior of hosts by a rich, cost-effective, and descriptive graph structure and attributes. Prior works use graphs with nodes identified by either only IP addresses [70, 75, 76, 214, 227] to discover the relationship across network hosts, or both IP addresses and port numbers (services) but simple edge attributes representing their connectivity [64]. To comprehensively characterize the network behavior of connected hosts (assets), I develop a 4-layer rooted-graph structure with nodes representing internal/external active hosts (IP addresses) and services (transport-layer port numbers) interconnected by attributed edges of inbound/outbound packets and flows. A total of 256 attributes are identified from this graph structure that collectively profile the behavior of connected hosts. I optimize my data structure and attributes by selecting 36 cost-effective yet predictive attributes and reducing the complexity of graph structure to two sub-graphs (each two-layer) to achieve both effective and scalable classification.

My second key novelty is the multi-grained inference scheme. Existing traffic classification methods focus on detecting certain applications (*e.g.,* video streaming [231]), discovering social relationships (*e.g.,* hosts that contact similar set of servers [215]), identifying certain popular host types (*e.g.,* website server, gaming server, P2P server [64]), or classifying host types that have fairly static and simple behavioral profiles (*e.g.,* IoT devices [47]). The inference scheme not only identifies known specific host types (N fine-grained classes) of an enterprise network, but also classifies non-typical unknown host types by six generic classes labeled by their dominant services. The coarse-grained model enables enterprises to detect emerging classes of assets on their network, potentially extending their fine-grained model. With this approach, every host is continuously classified, and the quality of network visibility can be dynamically adjusted by customizing the N-class fine-grained classifier.

Table 4.1: Summary of one-hour traffic trace data.

	TCP packets	**UDP packets**	**Internal hosts**	**External hosts**
In↓	908,836,429	225,435,536	217,708	462,573
Out↑	1,118,082,263	440,546,587	**21,258**	294,279

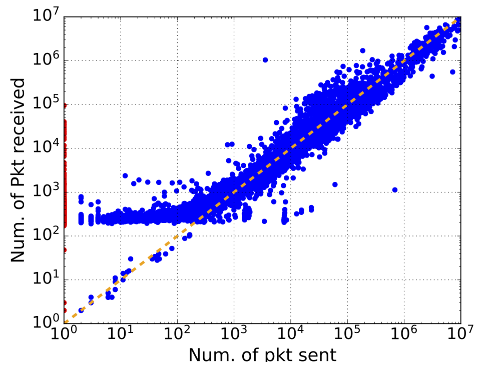
4.3 Understanding Network Behaviors of Enterprise Hosts

By analyzing one-hour traffic trace captured from two (one inbound and one outbound) 10 Gbps Internet border links, in this section, I: (a) highlight traffic statistics at the border of enterprise network in terms of distribution of inbound/outbound TCP/UDP traffic across the entire IP block of my organization and the variety of network services[1] (by transport-layer protocols and port numbers) on which internal hosts offer/access; (b) summarize network behavioral patterns of ten types of common enterprise networked assets (hosts) including website server, authoritative name server, VPN server, remote computing server, file storage server, email server, website proxy, recursive domain name resolver, and NAT gateway. In addition to these common types, there exist some non-typical assets with a diverse usage of custom transport-layer services. Therefore, I: (c) identify six aggregate classes of enterprise hosts based on their behavior regardless of the usage of transport services. These classes include TCP and UDP public-facing servers, TCP and UDP application proxies, NAT proxies, and end-hosts.

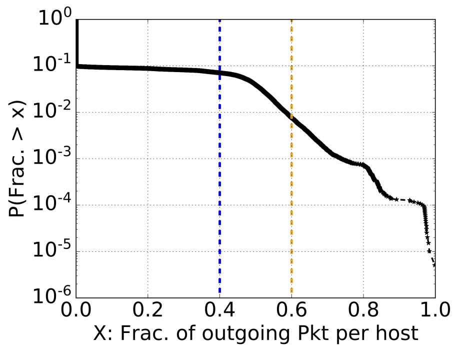
4.3.1 Overview of my PCAP Traces

To understand the behavioral profile of various enterprise hosts, I collected one-hour full traffic trace (inbound and outbound) from the two 10 Gbps Internet links outside

[1]I also use the term "transport service" to represent the combination of transport-layer protocol and port number, such as *TCP/443*.



(a) **in** versus **out** pkts per host.



(b) **out** fraction of pkts per host.

Figure 4.1: Dynamics of bidirectional traffic volume across internal hosts of the enterprise network: (a) count of incoming versus outgoing packets per host, and (b) CCDF of outgoing fraction of packets per host.

the border firewall of my university campus network. Appropriate ethics clearances[2] were obtained for this study.

Basic Statistics of my Dataset

My university owns three IPv4 blocks of size /16, giving a total of more than 196K public addresses. Using `tcpdump` tool, the first 96 bytes of all packets were recorded during the peak hour of a typical weekday (9-10am on 11 March 2019) – I verified that only 0.1% of packets were missed during this measurement. All headers of Ethernet, network, and transport layers are well-preserved, resulting in a total of 1.1 billion packets inbound and 1.6 billion packets outbound. The TCP/UDP composition of my dataset is shown in Table 4.1, and traffic rate of my collected data was about 10 Gigabits-per-second (Gbps) with an average 800K packets per second (pps).

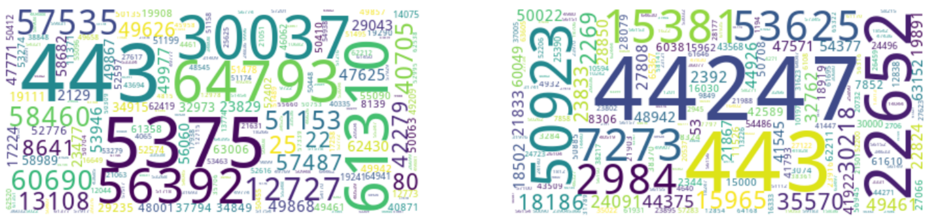
[2]UNSW Human Research Ethics Advisory Panel approval number HC17499, and CSIRO Data61 Ethics approval number 115/17.

Outgoing versus Incoming Traffic for Enterprise Hosts

Focusing on the hosts inside the enterprise network, as illustrated in Fig. 4.1(a), I observe that some hosts have almost equal amount of inbound and outbound packets (distributed across line y=x), whereas many other internal hosts display unbalanced behaviors. Also, there are "inactive" IP addresses (within the enterprise IP space) which receive packets from the Internet without sending any reply packet. For illustration purpose, I overlay inactive addresses by red dots along the y-axis in Fig. 4.1(a).

Fig. 4.1(b) shows the CCDF plot of the outgoing fraction of total packets per internal host. In this plot, I can see three main regions: outgoing fraction (*i.e.,* x-axis) less than 0.4, between 0.4 and 0.6, and more than 0.6, partitioned by vertical dashed lines (blue and orange). The first region, accounting for about 90% of internal IP addresses, represents internal assets which are either completely inactive (zero outgoing packet), or have much less number of outgoing packets than incoming – they seem to be target of scans or DoS attacks. The second region (*i.e.,* $0.4 \leq outFrac \leq 0.6$) represents the majority of active internal hosts which have almost same amount of incoming and outgoing packets – such behavior is often normal and expected. Lastly, the third region represents those enterprise hosts, each having a majority of their packets leave the network (*i.e.,* $outFrac > 0.6$) – seemingly participating in malicious activities (scans or DoS) which target external networks/hosts.

Note that a total of 217,708 enterprise hosts (*i.e.,* internal IP addresses) appeared in my dataset. However, only 21,258 of these addresses are active (*i.e.,* sending at least a packet to external entities). For the rest of my analysis, I produce a "cleaned dataset" by excluding packets of completely inactive hosts (those IP addresses that send no packet to the outside of the enterprise network but are probably the subject of incoming scans from the Internet). Therefore, my study will only be on those 21,258 enterprise hosts (IP addresses) with non-zero outgoing packets.



(a) TCP port numbers. (b) UDP port numbers.

Figure 4.2: Wordcloud of: (a) TCP, and (b) UDP, source port numbers in outgoing traffic of the enterprise network to the Internet.

Diversity in Transport-Layer Behavior

I now analyze the headers of outgoing packets sent by internal hosts, and zoom into their transport-layer services and source port numbers. This enables us to identify the role (*i.e.,* client or server?) of individual enterprise hosts at a high level, and infer the type of services (*e.g.,* HTTPS, SSH, or QUIC) they may offer. Similar insights can also be obtained from incoming packets and their destination port numbers – I omit this analysis to avoid redundant explanations.

To better visualize the diversity of transport-layer services, I use their word-cloud representation in Fig. 4.2. A weight is associated to each internal service (TCP/UDP port numbers) using the count of packets sent per service port number. It can be seen that most frequently used port numbers are either occupied by well-known services (ranging from 0 to 1023 [233] like *TCP/443* for HTTPS, assigned by the Internet standard RFCs) or certain de-facto servers (*e.g., UDP/443* used for Cisco VPN application), or randomly selected client ports (*e.g., UDP/44247* and *TCP/56392*) for a variety of network applications.

Let us have a closer look at some of these transport-layer ports which dominate my traffic trace. Considering *TCP/443*, highlighted by dark green in Fig. 4.2(a), I observe that about 500 enterprise hosts serve close to 20000 unique external hosts (IP addresses) by this top most popular web service. I manually verified (by performing reverse DNS lookup within the enterprise network) that 382 of these enterprise hosts are public-facing servers (such as the main student web-portal of university, VPN server, or web servers of various faculties/schools). Interestingly, *UDP/443*, highlighted by yellow in Fig.4.2(b), is also among popular UDP services provided by enterprise hosts to the public Internet – this port number corresponds to Cisco AnyConnect VPN service, handling more than 200 remote users during the one-hour traffic capture. Note that QUIC (developed by Google and predominantly used by Google servers) also operates on *UDP/443* – my university network, however, does not have any QUIC servers in operation.

Port numbers greater than 1023 are typically used on the client side (randomly chosen by operating systems) when attempting to contact their TCP/UDP servers. Interestingly, I found that *TCP/5375*, highlighted by dark purple in Fig.4.2(a), is the source port in about 9 million packets – almost all of these packets are sourced from an internal NAT gateway[3], having replied by approximately equal number of packets, which is an expected behavior. I examined the headers of packets sent/received by this NAT gateway, found that the vast majority (98.6%) of the packets sent are 60-byte TCP ACK packets in response to packets (average size of 117 bytes) received from an HTTPS (*i.e., TCP/443*) server operated by Microsoft Azure Could Computing Platform – probably a large download requested by an enterprise host behind the NAT gateway.

Similarly, *UDP/44247* is highlighted by dark blue in Fig.4.2(b) – 7 million packets generating a total volume of ≈10 GB to a Google cache server (operating on QUIC via *UDP/443*). This outgoing traffic was sourced from an internal host while its Google server consistently replied by small packets (of size 85 bytes on average) during this interaction – most likely, automatic sync with Google drive or uploading a video onto YouTube.

[3]I obtained the role of this host from its DNS name, consistent with the pattern of NAT gateway (will be discussed in §4.3.2)

As discussed above, the usage of network services (transport-layer port numbers) can be quite diverse, suggesting many types of roles (functionalities) across enterprise hosts. I, next, focus on popular host types that are commonly found in large typical enterprise networks.

4.3.2 Fine-grained Behavioral Profile of Enterprise Hosts

There are diverse types of hosts in an enterprise network that are trivial and not practical to enumerate. I now discuss ten popular fine-grained host types which are quite common in a large enterprise network (to prepare readers who may not be in this particular area), highlight their typical network behaviors using a rooted graph, and illustrate the existence of non-typical host types and behaviors.

Ten Popular Fine-grained Types of Enterprise Hosts

Website server is one of the most commonly used asset types that can be found in enterprise networks. These networked assets serve contents to public users via HTTP (*TCP/80*) or HTTPS (*TCP/443*). To retrieve enterprise web contents, external users initiate short TCP connections to these servers, sourced from randomly selected transport-layer services (*i.e.,* port numbers).

The second essential network asset for most of large enterprise networks is an **authoritative name server** which maps the organizational domain names to their respective IP addresses configured by the network administrator. This type of servers often operate on *UDP/53*, answering DNS queries from many external entities which use random source port numbers.

To enable their employees and users (*e.g.,* staff and students in case of university networks) who need to remotely access protected IT resources, enterprises often set up **VPN servers** (virtual private network servers). These servers provide tunneling connections (usually with longer duration) between remote endpoints and their internal networked resources.

Remote computing servers such as workstations and virtual machines, that provide enterprise staff with powerful resources to execute computationally-intensive tasks, are commonly used by research groups and departments. These networked assets typically offer remote accessing services like SSH (*TCP/22*) and Telnet (*TCP/23*).

Enterprises often need to store and manage their business-critical information, and make it available to their trusted departments and individual employees. **File storage servers** are therefore configured centrally and/or by sub-departments, so that staff and trusted entities can upload or access such critical data via bulk-transfer protocols like FTP (*TCP/21*).

Large organizations like an university may host their own email domains. **Mail servers** which are authoritative for email domains and handle the delivery of emails, are another common host type in an enterprise. They obviously operate on Email-related protocols such as SMTP (*TCP/25*, *TCP/465*, *TCP/587*) and others.

To secure and facilitate DNS lookups from enterprise hosts to public resolvers, large enterprises often have central and/or department-level **DNS proxies** configured, that only send DNS queries (sourced from random UDP ports) to external resolvers that listening on *UDP/53*.

Similarly, **website proxies** may also get configured to perform web lookups on behalf of enterprise regular end-hosts. These proxies use random source TCP port numbers to retrieve contents using short connections from Internet-based website servers that offer HTTP (*TCP/80*) or HTTPS (*TCP/443*) services.

In addition to application-specific proxies, **NAT gateways** [234] are usually configured as agents to provide outbound Internet connectivity (TCP/UDP) for hosts (often WiFi-connected devices) without public IP addresses, protecting them

Table 4.2: Ten popular host types identified from DNS names and their high-level network behavior.

Type	# hosts	sample DNS name	# internal services	# external services	flow duration	pkt. size
Website srv	61	*www.unswlawjournal.unsw.edu.au*	small, fixed	large, random	short	medium
Authoritative name srv	15	*ns1.sdn.unsw.edu.au*	small, fixed	large, random	short	small
VPN srv	13	*securevpn.nida.edu.au*	small, fixed	large, random	long	medium
Remote computer srv	16	*analyticalcentre2.chem.unsw.edu.au*	medium, fixed	large, random	long	small
File storage srv	14	*files.be.unsw.edu.au*	small, fixed	large, random	medium	large
Mail srv	18	*smtp.garvan.unsw.edu.au*	medium, fixed	large, random	short	medium
DNS proxy	7	*ns6.unsw.edu.au*	large, random	small, fixed	short	small
Web proxy	4	*wwwproxy2.library.unsw.edu.au*	large, random	small, fixed	short	medium
NAT gateway	256	*uniwide-pat-pool-a-b-c-d.gw.unsw.edu.au*	large,random	large,random	medium	medium
End-host	1961	*minzhaos-macbook-pro.ad.unsw.edu.au*	medium, random	small, random	medium	medium

from unsolicited incoming connections from the Internet.

Lastly, some enterprises may allocate certain **end-hosts** with their organizational public IP addresses, allowing them to directly communicate with the Internet. In my university network, machines connecting via Ethernet cable to wall ports in staff offices and certain labs will get public IP addresses.

In addition to these popular asset types (discussed above), I note that some other hosts like SNMP agents, video conferencing hubs, and Key management servers are not necessarily common in every enterprise network, hence not explicitly studied in this section. I will later in this section (§4.3.3) analyze the behavior of enterprise hosts by broadly considering their network activity over transport-layer services.

Inferring Host Types from Their DNS Name

DNS names associated with enterprise hosts can be helpful to some extent for inferencing their roles. In my 1-hour PCAP dataset, I extracted the domain name of each enterprise host (IP address) from outgoing DNS responses captured on the same day[4]. I managed to obtain the DNS name for 11,039 out of the 21,258 active enterprise hosts – more than 50% are found with a corresponding DNS name. However, by analyzing their DNS name (a combination of automatic string search and manual inspection), one may identify the role (type) of only 2,365 hosts (11%)

[4]I cross-checked against a separate dataset of daily DNS packets (incoming/outgoing), recorded at the border of my university campus network.

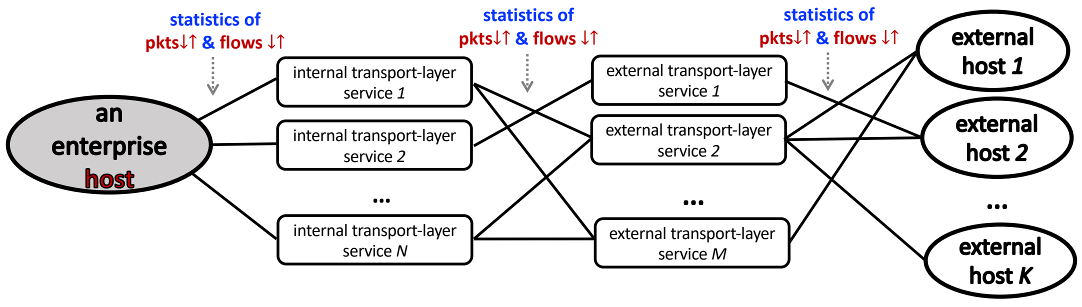

statistics of
pkts↓↑ & flows ↓↑
statistics of
pkts↓↑ & flows ↓↑
statistics of
pkts↓↑ & flows ↓↑
an enterprise host
internal transport-layer service 1
internal transport-layer service 2
...
internal transport-layer service N
external transport-layer service 1
external transport-layer service 2
...
external transport-layer service M
external host 1
external host 2
...
external host K

Figure 4.3: Rooted graph structure visualizes the network behavior of an enterprise host.

– only those servers and proxies that are directly managed by my university central IT department, public servers operated by certain departments/divisions (*e.g.*, a journal website maintained by the Law School of my university, as shown in the top row of Table 4.2), and NAT gateways or end-host devices that have names with identifiable patterns. For example, website servers usually use "`www`" in their name prefixes; some authoritative name servers in my university network have their names starting with "`ns`" followed by a number less than 5; the domain names of some DNS proxies contain "`ns`" followed by a number larger than 5; and NAT gateways configured by my IT department would have names with a certain pattern like "`uniwide-pat-pool-a-b-c-d.gw.unsw.edu.au`", given their static IP address is "`a.b.c.d`". The second and third columns of Table 4.2 summarize the count of hosts and a sample of their DNS names across the ten popular types.

I note that though DNS name may be used to label (identify the role of) some of the active hosts (mostly those which the central IT department manages), a large fraction of enterprise hosts (especially those which are managed by subdivisions or departments) do not have an identifiable name, hence remain unclassified. Therefore, a more comprehensive approach is required to profile the behavior of active hosts and identify their functionalities.

Profiling Network Behavior of Enterprise Hosts using Rooted Graph Structure

I profile the behavioral characteristics of each enterprise host by a rooted graph structure shown in Fig. 4.3, illustrating the communications between the enterprise host and external entities. As shown in this graph, the enterprise host (the leftmost node in Fig. 4.3) connects with external hosts (the rightmost nodes) via intermediate nodes (internal and external transport-layer services) and their corresponding edges. Edges on the graph represent statistical attributes (*i.e.,* count, size, and direction of packets; count, rate, volume, direction, and duration of flows) for unidirectional traffic exchanged across transport-layer services used by the enterprise host and corresponding external hosts. The direction of a flow is determined by the (internal or external) host which initiates the flow – for an "inbound" flow, the first packet is sent by an external host to the internal host. Thus, for every flow, I model inbound and outbound packets by separate edges.

I now generate and analyze the network behavioral profile of those popular host types (as ground-truth data) are identified by their DNS name. The last four columns in Table 4.2 briefly summarize the graph pattern (behavior) of each host type based on the features of nodes (*i.e.,* internal and external transport-layer services) and edges (*i.e.,* flow duration, and packet sizes).

Generally speaking, I observe both distinct and common behaviors across different host types. For example, servers like website server, authoritative name server, and remote computing server display a relatively focused set of internal transport services and fairly spread/broad set of random external transport services, while proxies (web and DNS) exhibit an opposite behavior – a wide range of random internal services and a narrow set of fixed external services). Furthermore, in terms of flow and packet characteristics, I observe some distinct patterns. For example, VPN servers typically maintain long flows with medium-size packets, while file storage servers generate medium-duration flows carrying large packets. To make my discussion more concrete, I next zoom into the behavioral profile of three representative host types, including website servers, web proxies, and NAT gateways.

Website servers: In summary, such servers are likely to offer a small set of internal TCP services to a wide range of external user TCP ports. The top internal services (by either packet or flow count) in both directions are likely to be *TCP/443* and *TCP/80*, while external port numbers are pretty random, distributed relatively evenly. Also, compared to other types of servers that predominantly operate over TCP services, website servers often maintain short flows with less than a few seconds.

Let us take a closer look at the behavior of an example website server (*i.e.,* student portal of my university), captured in my 1-hour PCAP trace. Starting with high-level observations on its packet-level characteristics, I found that the packet count is almost equal in both directions (*i.e.,* 917K for inbound and 913K for outbound). In contrast, the average size of outbound packets (245 Bytes) is more significant than that of inbound packets (62 Bytes). In terms of flow-level characteristics, the server received far more flows from the Internet (18K inbound flows) than it initiated towards external hosts (1K outbound flows). Also, the average duration of inbound flows is larger than that of outbound flows (*i.e.,* 1.4s and 0.2s, respectively). The server is found to use a total of 458 internal services (*i.e.,* ports), while about half of them (209 transport services) only appear in the inbound packets. Apparently, it is the victim of unsolicited traffic from the Internet. I further investigated those remaining 247 services (excluding unsolicited ones), and ranked them by their contribution to the number of packets and flows in each direction. Table 4.3 shows the top five services of the website server by four traffic characteristics (inbound/outbound packets and flows). The number in square brackets highlights the contribution of the corresponding service. Blue cells indicate services that are common among the top-5 across the four columns, while red cells indicate uncommon services which are not necessarily among the top-5 across all columns.

Table 4.3: Utilization of top-5 internal transport-layer services of a representative **website server**.

In↓ packets	**Out↑ packets**	**In↓ flows**	**Out↑ flows**
TCP/443 [97.2%]	*TCP/443* [98.1%]	*TCP/443* [73.0%]	*TCP/443* [52.0%]
TCP/80 [2.7%]	*TCP/80* [1.8%]	*TCP/80* [24.9%]	*TCP/80* [24.3%]
TCP/0 [0.0%]	*TCP/50923* [0.0%]	*TCP/50923* [0.9%]	*TCP/50923* [0.2%]
TCP/23 [0.0%]	*TCP/21631* [0.0%]	*TCP/21631* [0.0%]	*TCP/21631* [0.2%]
TCP/137 [0.0%]	*TCP/8641* [0.0%]	*TCP/8641* [0.0%]	*TCP/8641* [0.2%]

It is clearly seen that a vast majority of packets and flows in both directions are contributed by a small set of services that collectively characterize the behavior of this website server. I also analyzed external transport-layer services communicated with this asset[5]. I observed a wide range (more than 12K) of TCP services utilized fairly evenly – none of the top-5 external services contributed more than 0.5% across the four metrics of inbound/outbound packets and flows. Further, by analyzing the behavior of other website servers, I observed very similar patterns with slight variations in their use of transport-layer services *TCP/80* and *TCP/443*. Some use a mix of HTTP and HTTPS. In contrast, others prefer one of these TCP services.

Additionally, I note that website servers share certain network behaviors with public-serving assets like authoritative name servers, VPN servers, remote computing servers, and file storage servers. They all have a small set of internal services communicated with a wide range of external transport services. That said, each of these individual asset types is differentiated by their unique transport-layer services (*e.g., UDP/53* for authoritative name servers, or *TCP/22*, *TCP/23* or *TCP/3389* for remote computing servers). Also, they exhibit different characteristics of packets and flows (*e.g.,* VPN servers often maintain longer flows with an average duration of about a minute, and file storage servers typically use larger packets of average size over 350 Bytes).

Web proxies: These hosts use a wide range of random internal TCP ports, accessing *TCP/443* and *TCP/80* services offered by Internet servers.

[5]I omitted the table of results for external transport services.

Table 4.4: Utilization of top-5 internal transport-layer services of a representative **web proxy**.

In↓ packets	**Out↑ packets**	**In↓ flows**	**Out↑ flows**
TCP/39762 [3.6%]	*TCP/33280* [3.5%]	*TCP/3128* [0.2%]	*TCP/42108* [0.4%]
TCP/40576 [3.5%]	*TCP/41801* [2.6%]	*TCP/51771* [0.1%]	*TCP/51769* [0.4%]
TCP/58355 [3.4%]	*TCP/8513* [2.6%]	*TCP/2869* [0.1%]	*TCP/15833* [0.3%]
TCP/44379 [3.3%]	*TCP/40576* [2.4%]	*TCP/7253* [0.1%]	*TCP/3985* [0.3%]
TCP/53718 [3.2%]	*TCP/39762* [2.4%]	*TCP/48782* [0.1%]	*TCP/48471* [0.3%]

As an example, let us highlight the traffic profile of a web proxy in my university network. At a high level, this proxy exchanged an almost equal number of packets in each direction (1.2M outbound and 1.5M inbound), while it had far more outbound flows (36K) than inbound flows (3K). Table 4.4 provides relatively fine-grained insights into the behavior of this type of host by listing its internal top-5 transport-layer services along with their respective contribution to the host traffic at packet and flow levels. I can see that all cells in this table are highlighted by red color, suggesting uncommon services across the four metrics. Also, it is seen that these top services are utilized fairly evenly by individual columns of inbound/outbound packets and flows. As for external transport services (*i.e.,* the services on external hosts), instead, a vast majority of packets and flows (*i.e.,* more than 99%) are narrowly focused on *TCP/443* and *TCP/80.* I saw earlier this set of dominant services on the internal side of the website server.

Other application proxies in my dataset displayed similar behavioral patterns, except their usage of transport services – for example, DNS proxies primarily use UDP for their transport-layer protocols, with *UDP/53* dominating their external service.

NAT gateways: Unlike website servers and proxies described so far, NAT gateways use a much more random set as well as a broader range of transport services, both internally and externally, across TCP and/or UDP protocols. Unsurprisingly, they behave very much like clients that initialize flows towards external hosts.

For example, a WiFi access point (NAT gateway) on my campus network exchanged almost the same number of packets in each direction (*i.e.,* 2.2M inbound and 2.3M outbound) as captured by my 1-hour traffic trace. In terms of flows, I observed about 50K outbound flows and 30K inbound flows. This large count of incoming flows is unexpected. Further investigation revealed that 28K (91%) of the inbound flows are unsolicited and not responded. The responded incoming flows (remaining 2K) found during the first five minutes of my trace – probably they were initiated (as outbound) just before the commencement of my traffic capture. Note that I will discuss in §4.4 that why my final classifiers will be trained by attributes of outbound traffic of individual hosts.

Analyzing the internal transport-layer services of this network asset does not manifest any pattern (a wide range of seemingly random TCP/UDP port numbers). For external transport-layer services, instead, more than 99% of inbound/outbound packets and flows are contributed by top-5 services including HTTPS (*TCP/443*), QUIC (*UDP/443*), HTTP (*TCP/80*), DNS (*UDP/53*), and IMAPS (*TCP/993*), while strongly dominated (more than 90%) by HTTPS and HTTP.

Other NAT gateways are found to exhibit similar behavior with slight variation in their use of transport services. Note that end-hosts also share this behavior, but with lighter activity in traffic volumes and range of transport services.

Enterprise Hosts with Non-Standard Behavior

By further analysis of my PCAP dataset, I identified some other types of network assets that either fundamentally differ from typical hosts (Table 4.2) or display significantly new patterns in addition to the expected behavior of typical hosts.

Non-typical host types: Some enterprise hosts utilize transport-layer services that are less popular in a typical enterprise IT infrastructure. For example, one host has two internal TCP services (*TCP/636* for LDAP and *TCP/389* for LDAPS) that together contribute to more than 97% of outbound/inbound packets and flows. Also, I found another server with a distinct internal service *TCP/11371* for HKP (used by Key management servers). I verified that this host is operated by a non-profit organization within my university campus by inspecting its DNS name.

Non-standard variants of typical host types: Among the hosts labeled (by their DNS name) as one of the ten typical types, some display distinct behaviors (the use of additional services) compared to their respective cohort. A website server configured by a research group in an engineering department, has 5 distinct internal services, namely *TCP/443* and *TCP/80* (expected standard services), as well as *TCP/3306*, *TCP/2222*, and *TCP/23* that respectively correspond to MySQL, SFTP, and Telnet (non-standard services) – respectively, contributing to 28.1%, 24.6% 11.0%, 4.4%, and 3.6% of outbound packets. Indeed, running a server with multiple roles is not a best practice [235], since various services demand specific policies (for their particular vulnerabilities and risks) to be enforced at the network. Another example is a name server managed by an engineering department (not the central IT) in my university. I found this host to function as both authoritative name server and DNS proxy – *UDP/53* is the most dominant in both internal and external transport-layer services. In addition to a combined role, it is not a recommended security practice for an enterprise host to resolve DNS queries from the public Internet [236].

4.3.3 Coarse-grained Behavioral Profile of Enterprise Hosts at Transport-Layer

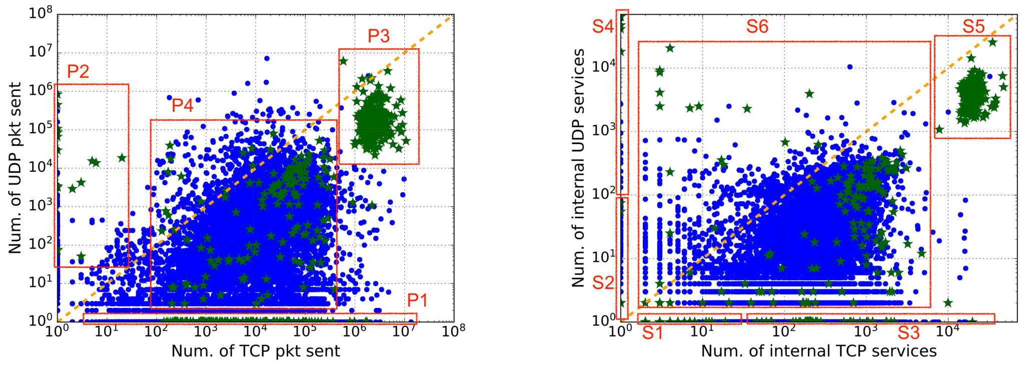
From what I have observed from my traffic traces, a growing set of profiles (classes) will exist for enterprise hosts, considering the specific transport-layer services ("fine-grained") they offer and/or consume. This makes it practically challenging to capture and maintain those individual classes for a real-time asset classification task, to function continuously. However, it is possible to cap the number of host classes by considering their "coarse-grained" behavior at transport-layer by aggregating their fine-grained transport services. In what follows, I discuss how enterprise hosts can be categorized under six coarse-grained types, including TCP-dominant server, UDP-dominant server, TCP-dominant proxy, UDP-dominant proxy, NAT gateway, and end-host.

Six Coarse-grained Host Types by Transport Services

Enterprise hosts that offer network services to users on the public Internet can be either a **TCP-dominant** or **UDP-dominant server**, depending on the distribution of transport-layer services (*i.e.,* TCP or UDP) in their network traffic. They expose a small set of internal transport-layer services (either TCP or UDP) contacted by a wide range of external transport services initiated by Internet clients. For instance, the website servers mentioned above primarily operate on HTTP (*TCP/80*) and/or HTTPS (*TCP/443*) and thus are TCP-dominant servers. An organizational VPN server can operate on both *TCP/443* and *UDP/443*. Still, a vast majority of packets and flows belong to TCP protocol, and thereby a TCP-dominant server.

Proxies that access certain services like Web (*TCP/443*) or DNS (*UDP/53*) on the Internet can be categorized as either **TCP-dominant proxy** or **UDP-dominant proxy**. They tend to use a highly diverse and random set of internal transport-layer services, either TCP or UDP. Also, proxies (depending upon their role) consume a narrow set of external transport services. For example, DNS proxies (as UDP-dominant proxies) use a wide range of internal UDP service numbers to send DNS query packets to external servers operating on the network service *UDP/53*.

Lastly, **NAT gateways** and **end-hosts** are relatively distinct by their use of a wide range of internal services, consuming a mix of external services over both TCP



(a) Packet sent. (b) Internal transport services.

Figure 4.4: UDP versus TCP outgoing traffic per internal host: (a) count of packets sent, and (b) unique internal transport-layer service used.

and UDP protocols.

Grouping Enterprise Hosts by Coarse-grained Behavior of Transport-layer

I now analyze the coarse-grained transport-layer behavior of all active enterprise hosts that appeared in my 1-hour cleaned dataset. I compute the number of packets, flows, internal transport services, and external transport services of both directions (TCP and UDP separately) for a given enterprise host. In what follows, I elaborate on the coarse-grained types (discussed above) by highlighting some of network behaviors for the hosts with ground-truth label.

Let us concentrate on the outgoing traffic of individual hosts and analyze their distribution of packets and services[6]. Fig. 4.4(a) is the scatter plot of UDP versus TCP packets per host. Fig. 4.4(b) displays UDP and TCP ports distributed across individual hosts. Each green star highlights a ground-truth host identified by their DNS names, and blue dots represent other hosts.

I observe that hosts (with ground-truth label) from various coarse-grained types

[6]I omit the insights obtained from inbound packets and flows for brevity.

display distinct behavior at least by the two distributions illustrated in Fig. 4.4. TCP-dominant servers appear on the narrow region $\boxed{\text{P1}}$ in Fig. 4.4(a) and narrow $\boxed{\text{S1}}$ in Fig. 4.4(b), indicating their heavy and concentrated activities via specific TCP services, that is, sending out many TCP packets through a limited set of TCP services. Similarly, UDP-dominant servers are expected to appear on regions $\boxed{\text{P2}}$ in Fig. 4.4(a) and $\boxed{\text{S2}}$ in Fig. 4.4(b). TCP-dominant proxies are located on $\boxed{\text{P1}}$ and $\boxed{\text{S3}}$ regions (large number of TCP packets, and heavily distributed in TCP services), while UDP-dominant proxies sit on the regions of $\boxed{\text{P2}}$ and $\boxed{\text{S4}}$ in the two scatter plots. NAT gateways, given they represent a large number of end-hosts, appear in the upper right corner of both figures (*i.e.,* $\boxed{\text{P3}}$ and $\boxed{\text{S5}}$). Lastly, end-hosts mainly fall in two broader regions ($\boxed{\text{P4}}$ and $\boxed{\text{S6}}$) for their diverse and less-concentrated use of TCP and UDP transport-layer services.

4.4 Classifying Enterprise Hosts

In this section, I develop a multi-grained classification scheme that classifies enterprise hosts into ten popular fine-grained types (discussed in §4.3.2) and six aggregated coarse-grained types (discussed in §4.3.3) with highlights of their dominate services at transport-layer. I discuss host-specific traffic attributes used as inputs of my models, and quantify their importance, independence, and computational cost. Next, I train, tune, and validate two multi-class ML models (as my baseline models) for both fine-grained and coarse-grained classifications. I enhance the practicality of my method by judiciously selecting subsets of attributes and adjusting retention duration to enable it for real-time operation at scale. Their classification performance is compared with that of my baseline models. Lastly, I quantify the efficacy of my inference scheme by applying it to a fresh set of traffic instances not seen during the training phase of my ML classifiers.

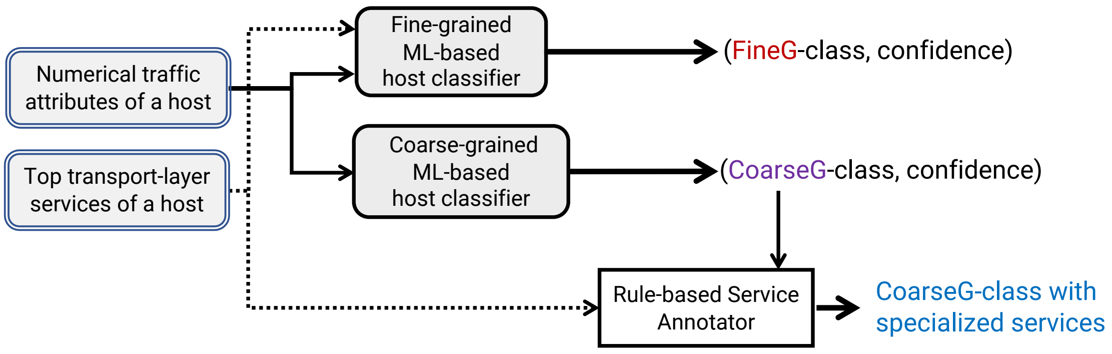

Numerical traffic attributes of a host
Top transport-layer services of a host
Fine-grained ML-based host classifier
Coarse-grained ML-based host classifier
(FineG-class, confidence)
(CoarseG-class, confidence)
Rule-based Service Annotator
CoarseG-class with specialized services

Figure 4.5: My multi-grained classification scheme.

4.4.1 Multi-Grained Classification Scheme

Ideally speaking, every enterprise host is expected to be labeled by their fine-grained type (*e.g.,* website server, authoritative name server, or web proxy). However, it becomes a challenging task in practice when many operational hosts may not display a narrowly identified behavior (using less-common network services or having mixed roles). Therefore, those hosts can at least be classified as one of the six coarse-grained types (*e.g.,* TCP-dominate server or TCP-dominate proxy) by aggregating transport-layer services.

Therefore, my classification scheme infers the type of enterprise hosts at two levels of granularity, namely fine-grained and coarse-grained. Fig. 4.5 illustrates the structure of my scheme where I take two groups of statistics computed from the rooted graph of each host (§4.3.2) as inputs. The first group pertains to "numerical traffic attributes" that describe the activity behavior of the host without inclusion of specific transport-layer service name (*e.g.,* "*TCP/443*"), while the second group highlights "top transport-layer services" (internal and external) of the host. My classification scheme contains three functional modules. The fine-grained model is a N-class classifier that receives both input groups and generates a fine-grained class (*e.g.,* website server) a confidence value for the input host. For the use-case of my campus network, I considered ten popular host types ($N = 10$). I note that various enterprises may want to customize the number of fine-grained classes (extending my ten classes or choosing a subset of these ten classes) depending on their preference and composition of their network. The other two modules are designed for coarse-grained inference. A six-class classifier model gives an intermediate prediction (*e.g.,* TCP-dominate server) with a confidence level by processing numerical traffic attributes of a host. A rule-based annotator will take the intermediate prediction as well as top transport-layer services of the host to generate the final coarse-grained host type (*e.g.,* "a TCP-dominate server by *TCP/443* and *TCP/80*").The rule-based annotator uses my insights obtained in §4.3.3 to annotate servers by their top internal services, proxies by their top external services. NAT gateways and end-hosts are not annotated. It is important to note that the coarse-grained inference model is generic, and hence applicable to any enterprise setting. When the prediction is annotated, it enables the network operator to discover emerging asset classes, potentially extending the fine-grained model by new classes (if desirable).

Each of the two predictions (fine-grained and coarse-grained) will contain a class label and confidence level, helping network operators better manage their assets by choosing certain inference results. A coarse-grained label is accompanied by top transport-layer services per host, providing additional information for further investigation post automatic inference. I note that various enterprises may want to customize the number of fine-grained classes (extending my ten classes or choosing a subset of these ten classes) depending on the variety of asset types they may have on their network. Coarse-grained classes, on the other hand, are generic to any enterprise setting.

4.4.2 Attributes of Host Network Behavior

I now discuss and evaluate my host-specific attributes required for the inputs of my multi-grained host behavior inference scheme.

Attributes

Having understood various host profiles in §4.3, I identify a set of attributes that are computed from the rooted graph of an enterprise host (shown in Fig. 4.3) to capture their comprehensive network behavior. Given nodes (enterprise host, internal transport-layer services, and external transport-layer services) in Fig. 4.3, I extract a total of 256 attributes, including 176 numerical traffic attributes and 80 categorical attributes indicating top transport-layer services. To better describe them, I choose the name of each attribute according to a pattern "*metricType-direction-resolution*", where "*metricType*" capture statistical measures of traffic volume and utilization of transport services, "*direction*" highlights inbound↓ vs. outbound↑, and "*resolution*" is packet-level or flow-level. In what follows, I discuss three groups of numerical attributes, namely aggregate host activity, utilization of internal transport-layer services, and utilization of external transport-layer services), along with categorical attributes (*i.e.,* top transport-layer services).

Aggregate host activity: The leftmost node in Fig. 4.3 is the enterprise host with edges of inbound and outbound packets/flows representing its aggregate network activity. I use two metrics *AvgSize* and *VarSize* to highlight the average size and variance of the traffic units (packet-level or flow-level) for a given host.

My analysis in §4.3 revealed that various enterprise hosts could exhibit different traffic distribution in each direction (*i.e.,* inbound↓ or outbound↑) across resolutions (*i.e.,* packet or flow). Therefore, I compute the above two metrics for both directions and both resolutions, resulting in 8 attributes for this group. An example of these attributes is *AvgSize-↓-Pkt*, indicating the average size of inbound packets.

Utilization of internal transport services: Moving to attributes of the second leftmost node in Fig. 4.3, I find various services via protocol type TCP or UDP, distributed across inbound and outbound directions. For this group of features, I identify 21 statistical metrics.

I start by two aggregate metrics, namely *DominTypeInSrv* (*i.e.,* the dominant protocol type of enterprise services by packet/flow count), and *FracMinorTypeInSrv* (*i.e.,* the fraction of packet/flow count associated with internal services of the protocol type at minority).

To capture the distribution of host traffic across internal services, I consider two ways, namely (i) packet/flow count, and (ii) unique external service count, for ranking their individual contributions. **First**, by considering the total count of packets/flows, I identify nine statistical metrics. Given a list of services ranked by their packet/flow count from largest to smallest, I compute: (a) traffic fraction of the top service (highest activity), the first quartile service, the second quartile service and the third quartile service, denoted by *FracTopInSrv*, *FracQ1InSrv*, *FracQ2InSrv*, and *FracQ3InSrv*, respectively; (b) the variance of traffic fraction across internal services denoted by *VarInSrv*; (c) fraction of internal services above average, above average plus one-sigma, above average plus two-sigma, and above average plus three-sigma, denoted by *FracAbvAvgInSrv*, *FracAbvAvg1SigInSrv*, *FracAbvAvg2SigInSrv*, and *FracAbvAvg3SigInSrv*, respectively. **Second**, by considering the count of corresponding unique external services, I identify ten statistical metrics. Nine of the metrics are computed in the same way as described above. Additionally, I use the ratio of internal service count and external service count, denoted by *RatioIntExtSrv*.

For this group, I identify a total of 21 metrics across two directions and two resolutions, resulting in 84 numerical attributes.

Utilization of external transport services: Similar to the characterization of internal services (discussed above), I capture the distribution of host traffic across external services by 21 metrics (each with two directions and two resolutions), resulting in a total of 84 attributes. For brevity, I omit details of attributes.

Top transport-layer services: As discussed in §4.3.2, various host types may focus on certain transport-layer services, appeared as dominant internal and/or external services. I use the top five internal and external services ranked by certain properties of edges in the graph depicted in Fig. 4.3. Top-ranked internal services are determined by: (i) the volume of traffic (packet, flow) in each direction (outbound, inbound) on the left edges connecting them to the enterprise host, and (ii) the count of non-zero (inbound/outbound/ packet/flow) middle edges connecting them to the external services. Similarly, top-ranked external services are determined by: (i) the volume of traffic on the right edges connecting them to external hosts, and (ii) the count of non-zero middle edges connecting them to the internal services. For example, the attribute *TopInSrvVol-*$\uparrow$*-Pkt* indicates the top internal transport service ranked by the volume of outbound packets, while *TopInSrvNZExSrv-*$\uparrow$*-Pkt* is ranked by the count of external services with non-zero edges of outbound packets. As a result, 80 attributes are identified. I note that this group of attributes are categorical and can not directly be fed to numerical machine learning models. I use a method called integer encoding that maps categorical attributes to numerical values to address this issue. This lightweight method is suitable for a wide range of categorical values [237] compared to its alternatives like one-hot encoding. It can also be handled well by tree-based classifiers.

Dataset Preparation

I compute attributes of hosts with the ground-truth label (fine-grained and coarse-grained types) identified by DNS names in §4.3.2. A run-time rooted graph for each of those hosts is tracked with a retention duration of 1 hour (*i.e.,* edges that are inactive for more than 1 hour will get removed), and host attributes are calculated every minute. A dataset consisting of 928,946 records is developed from 24 hours (between 11am on 31st May 2019 and 11am on 1st June 2019) worth of traffic traces. The number of instances for each host type is balanced through resampling, which is a common strategy to handle imbalanced dataset.

Merit of Attributes

I now evaluate the merit of individual attributes (a total of 256 attributes: 8 for aggregate host activity, 84 for utilization of internal transport service, 84 for utilization of external transport service, and 80 for top transport-layer services) in predicting the type of their corresponding host. I also quantify the dependency between each pair of attributes and the cost for computing each attribute in real-time. The insights gained from this section will guide us (later in §4.4.3) to balance the cost against the accuracy of my ML classifiers.

Importance: To quantitatively justify the efficacy of my identified attributes, I use "information gain" to measure their importance. *InfoGain* of an attribute indicates how much information the attribute provides with respect to the classification goal.

$$\text{InfoGain}(\mathbf{a}) = \text{Entropy}(\mathbf{h}) - \text{Entropy}(\mathbf{h}|\mathbf{a}) \tag{4.1}$$

$$\text{Entropy}(\mathbf{h}) = -\sum_{h \in \mathbf{h}} p(h) \log_2 p(h) \tag{4.2}$$

$$\text{Entropy}(\mathbf{h}|\mathbf{a}) = -\sum_{a \in \mathbf{a}} p(a) \sum_{h \in \mathbf{h}} p(h|a) \log_2 p(h|a) \tag{4.3}$$

As shown in Eq. 4.1, *InfoGain* [238] of attribute a is derived as the difference between the entropy of original (unsorted) host type (*i.e.,* $\boldsymbol{h}$) (Eq. 4.2) and the entropy of host type sorted by the attribute $\boldsymbol{a}$ (Eq. 4.3).

Normalized importance of each attribute is calculated and labeled by their direction (*i.e.,* inbound or outbound), resolution (*i.e.,* packets or flows), and associated node (*i.e.,* internal or external transport-layer services). Fig. 4.6 summarizes the importance of attributes across these three perspectives as box plots where each box highlights a range from the first quartile to the third quartile of merit values within their respective group. It can be seen that all attributes contain some information

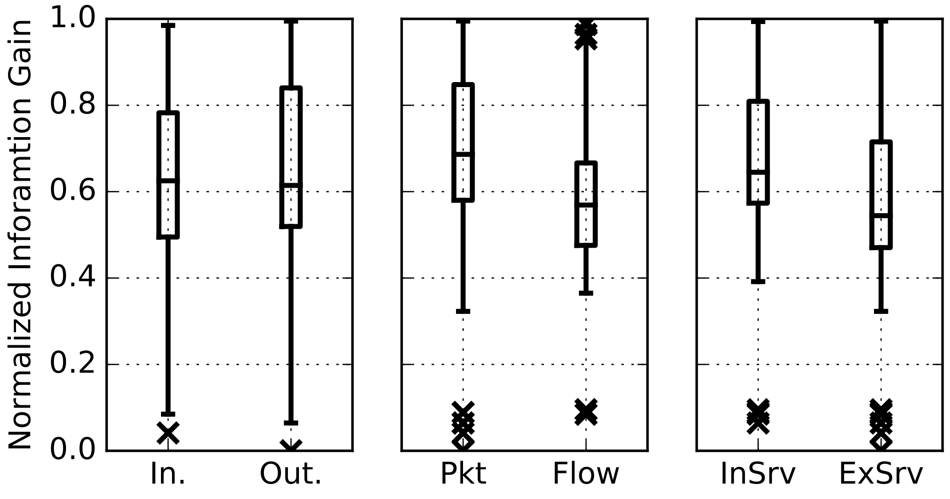


Figure 4.6: The merit of attributes across different perspectives.

for predicting the type of hosts. Only a few attributes display a low importance (InfoGain values smaller than 0.2) – these attributes are primarily pertinent to the protocol type of transport layer (*e.g., DominTypeInSrv-↓-Pkt*). As indicated by the leftmost subplot in Fig. 4.6, outbound attributes are slightly more predictive than their inbound counterparts – possibly because inbound traffic may contain noises such as scans or unsolicited traffic, providing information not indicative of the role of internal hosts. Moving to the middle subplot, packet-level attributes (given their higher resolution) relatively outweigh flow-level ones. Lastly, the rightmost subplot shows that attributes pertinent to internal transport services yield a higher power in predicting the type of enterprise hosts than external services.

Independence: Certain attributes may positively or negatively correlate with others. To quantify the correlation between a pair of attributes, I use the projection coefficient p calculated by Eq. 4.4, where $\mathbf{a1}$ and $\mathbf{a2}$ are the two attribute arrays, each containing a set of instances (*e.g.,* $a1_0$ and $a2_0$). As an simple example, if I have three hosts with their two attribute values as $[a1_0, a2_0]$, $[a1_1, a2_1]$, and $[a1_2, a2_2]$, the two attribute arrays $\mathbf{a1}$ and $\mathbf{a2}$ are then written as $[a1_0, a1_1, a1_2]$ and $[a2_0, a2_1, a2_2]$, respectively.

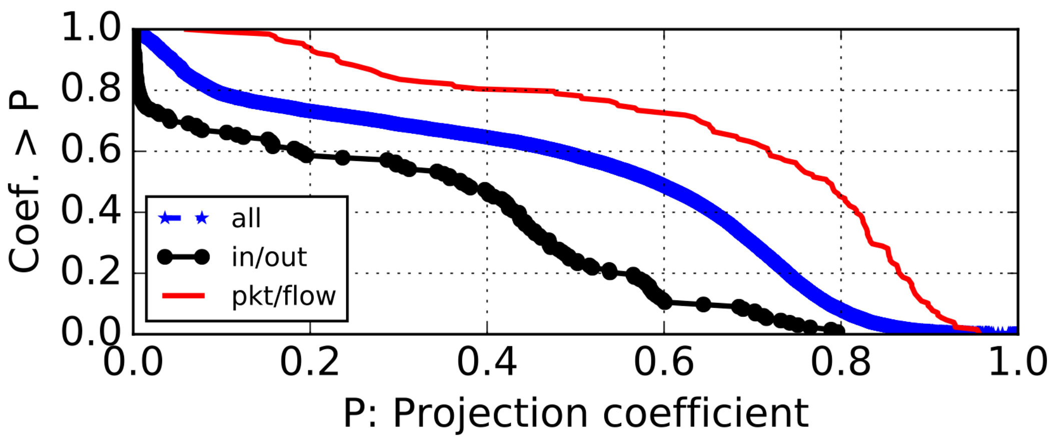


Figure 4.7: Correlation of attribute pairs.

$$p = \frac{\mathbf{a1}\ \mathbf{a2}}{|\mathbf{a1}||\mathbf{a2}|},\ \ \mathbf{a1} = [a1_0, a1_1, .., a1_n], \mathbf{a2} = [a2_0, a2_1, .., a2_n] \tag{4.4}$$

Fig. 4.7 shows the CCDF plot of the projection coefficient across all $\binom{256}{2} = 32640$ attribute pairs (shown by blue dots), 128 corresponding pairs in two directions like *FracTopInSrv-↓-Pkt* versus *FracTopInSrv-↑-Pkt* (shown by black dots), and 128 corresponding pairs in two resolutions like *FracTopInSrv-↓-Flow* versus *FracTopInSrv-↓-Pkt* (shown by red dots). I observe that packet-based attributes are largely correlated with their corresponding flow-based attributes – 70% of pairs on the red curve display a correlation value greater than 0.6. On the other hand, outbound attributes are loosely correlated with their inbound counterparts. Almost 80% of pairs on the black curve display a correlation value smaller than 0.5 – probably because inbound traffic is relatively polluted by unsolicited traffic.

Computational cost: To compute the attributes of an enterprise host in real-time, I need to continuously maintain and update their data structure with run-time statistics. Therefore, I use the complexity of the data structure required to obtain an attribute as a proxy of its cost.

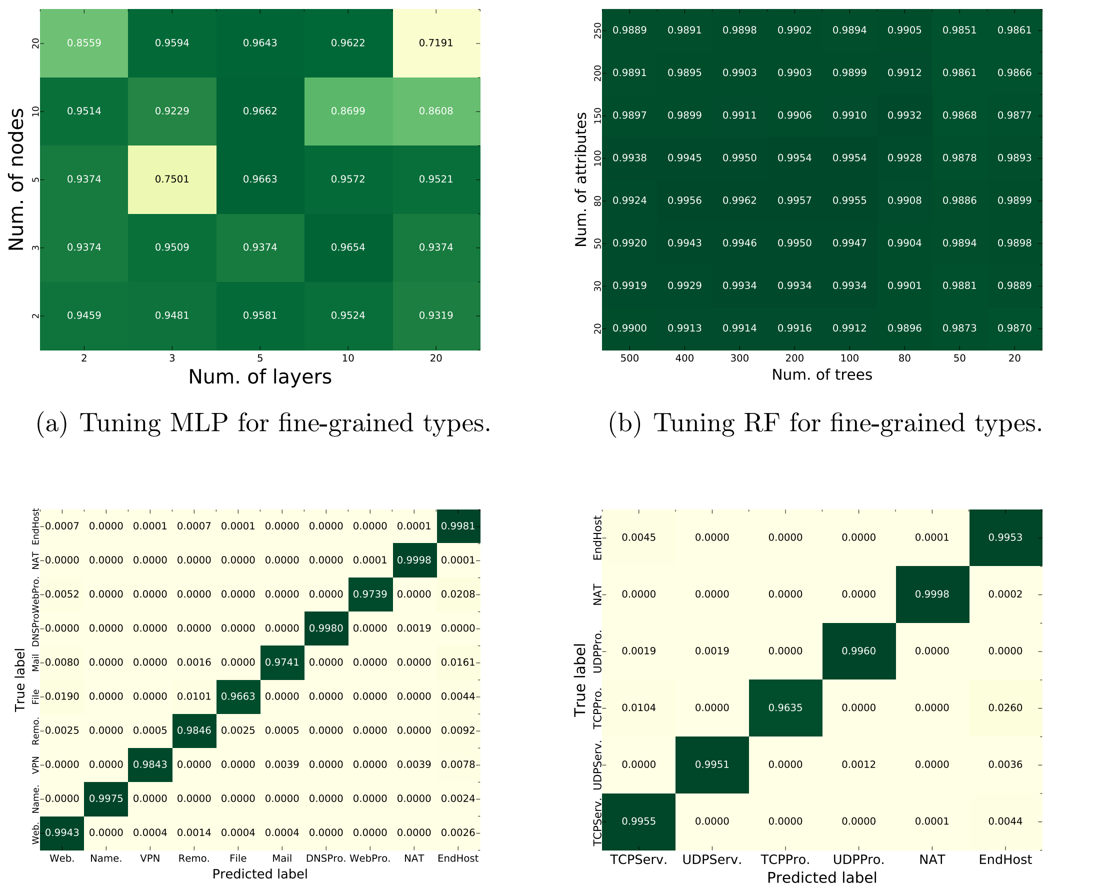
As illustrated by Fig. 4.3, a fine-grained graph contains four key layers including enterprise hosts, internal transport-layer services, external transport-layer services, and external hosts. Flow-based attributes (*e.g., AvgSize-↑-Flow*) require all four key layers of metadata (*i.e.,* enterprise host, internal transport-layer service, external transport-layer service, external hosts); hence, they are computationally heavier. Attributes like those that pertain to both internal and external transport services (*e.g., TopInSrvNZExSrv-↑-Pkt*) need three key layers of metadata (*i.e.,* enterprise host, internal transport-layer service, external transport-layer service). Lastly, packet-based attributes may only need one or two layers of metadata (*e.g.,* enterprise host for *AvgSize-↑-Pkt*; enterprise host and internal transport-layer service for *FracTopInSrv-↓-Pkt*); hence, they are computationally lighter.

I, therefore, associate qualitative costs of the low, medium, high, and ultra-high to attributes that require one, two, three, and four key layers of metadata, respectively. As a result, of the 256 attributes, 4 are low cost, 68 are medium cost, 56 are high cost, and 128 are ultra-high cost.

Maintaining host attributes can be practically challenging at scale, particularly for the high traffic rates of a large enterprise network. Therefore, one may choose to focus on a subset of attributes that give more predictive power, are independent, and incur a reasonable cost. For example, attributes of outbound packets that carry significant information and can be computed at low/medium cost.

4.4.3 Training, Tuning, and Cross-Validating Classifiers

I train and evaluate my ML classifiers for both fine-grained and coarse-grained host types using two famous algorithms, namely multi-layer perceptron (*i.e.,* MLP, a neural network algorithm) and Random Forest (a collection of decision trees). My models are trained and cross-validated using my ground-truth dataset (discussed in §4.4.2). Parameters of each model are tuned to achieve their best performance. I also train models with subsets of attributes considering computing costs and compare



(a) Tuning MLP for fine-grained types. (b) Tuning RF for fine-grained types.

(c) The best RF model for fine-grained types. (d) The best RF model for coarse-grained types.

Figure 4.8: Performance of models: (a) tuning parameters of RF models for host types, (b) accuracy of the best RF model for fine-grained host classification, and (c) accuracy of the best RF model for coarse-grained host classification.

their accuracy with that of best performing models.

Performance of Models

I now describe the training and validation of my classification models for predicting fine-grained and coarse-grained host types. My classifiers are generated using two popular algorithms: multilayer perceptron (MLP) and random forest (RF). MLP classifiers are tuned by varying the number of layers and the number of nodes in each layer. Random forest algorithms are tuned by varying the number of trees and the number of attributes for each tree. As a visual illustration, the accuracy (ranging from 0 to 1) of MLP and RF classifiers for host types are shown in Fig. 4.8(a) and 4.8(b), respectively. The best performance of the fine-grained classifiers is 84.69% (MLP) and 99.62% (RF). For the coarse-grained classifiers, the best performance for MLP and RF are 98.15% and 99.72%. It is clear that my RF models outperform the MLP models at both angularities. I, therefore, use the two best RF models as a baseline for the rest of this chapter.

Let us now zoom into the accuracy of my baseline models in predicting the type of hosts. I observe in Fig. 4.8(c) that more than 96% of fine-grained instances are correctly classified. The performance is higher (more than 99%) in certain classes such as website servers and authoritative name servers. I note that 1.90% of file server instances are misclassified as website servers. Also, 2.08% of web proxy instances are misclassified as end-hosts. Moving to the coarse-grained classifier in Fig. 4.8(d), almost all classes receive an accuracy of more than 99%, while 2.60% of TCP proxy instances are misclassified as end-hosts.

Confidence Levels

My models output a measure of confidence (a value between 0 and 1) with each prediction. Let us start with the fine-grained host type classification whereby every predicted instance (correct and incorrect) is accompanied by a confidence level of more than 0.60, while correctly classified instances come with fairly higher confidence greater than 0.85. A fraction (9%) of misclassified instances receive a confidence level of more than 0.80 (relatively high). By analyzing their attributes, I found that affected hosts indeed displayed a different network behavior other than their expected type. For example, a NAT gateway is classified as a web proxy with an 0.87 confidence. This host only had a small number of TCP flows destined to external services *TCP/443* and *TCP/80*, representing the typical behavior of a low-profile web proxy. The majority of misclassified fine-grained instances carry a confidence level lower than 0.80. They are likely to be associated with non-typical behavioral patterns; hence, they require further analysis such as inference by the coarse-grained classifier, annotation of their popular transport-layer services, or deeper pack-based investigation. As another example, a website server is classified as end-host with low confidence of 0.61. After looking into its attributes, I found that it holds mixed functionalities including DNS, remote accessing, file storage, and accessing external servers via an extensive range of internal transport-layer services. I made similar observations with the coarse-grained classifier.

Recall that my multi-grained classification scheme (§4.4.1) classifies a given enterprise host into ten fine-grained types and six coarse-grained types. When the confidence of the fine-grained model is not high enough, network operators may choose to resort to the prediction of the coarse-grained model. However, the coarse-grained classifiers may still give a low-confidence prediction, requiring further investigations. In my prototype deployment (will be described in §4.5), enterprise hosts that receive a low confidence level[7] from the coarse-grained model will be isolated for deeper packet-level investigation.

ML Classifiers with Partial Information

I now analyze the performance of my models trained on subsets of the 256 attributes considering computing costs and retention period, making them more suitable and scalable for real-time operation in large enterprise networks.

Optimizing attribute selection: I first train and tune RF models for both classification tasks using various combinations of attributes considering their qualitative costs. Table 4.5 summarizes the performance of best performing fine-grained models at various configurations – I have omitted results of the coarse-grained models as similar observations were made for both types of models. Note that the number of attributes in hyper parameters are not previously defined and randomly selected

[7]I set the confidence threshold to 0.80 for both models.

Table 4.5: Fine-grained models using subsets of attributes.

Configuration	**Best model**	**Hyper parameters**
Full attribute set	99.62%	80 attributes, 300 trees
Outbound↑ only	99.51%	50 attributes, 200 trees
Inbound↓ only	83.27%	50 attributes, 150 trees
↑ excluding ultra-high costs	99.42%	50 attributes, 200 trees
↑ low & med. costs only	**98.88%**	**30 attributes, 200 trees**
↑ low costs only	35.32%	5 attributes, 50 trees

by the RF training algorithm.

The baseline model with 99.62% accuracy (top row in Table 4.5) is trained on the full set of attributes (§4.4.3). Interestingly, only outbound attributes (half of the attributes) yield a model with a very similar accuracy of 99.51% (second row). However, inbound half of the attributes cannot achieve better than 83.27% accuracy (third row). This is probably because outbound traffic is less polluted than inbound. Focusing on the outbound traffic, I can still achieve fairly high accuracy of 99.42% if I exclude ultra-high attributes (fourth row). The overall accuracy is slightly compromised to 98.88% by considering only low and medium-cost attributes of outbound traffic (fifth row). However, as highlighted by the sixth row, I cannot further optimize the cost when only low-cost attributes are used to train the model since the obtained accuracy is far unacceptable. Therefore, I choose to continue with the best models trained on low-cost and medium-cost attributes of outbound traffic, given a combination of performance and cost metrics.

Tuning retention period: As mentioned earlier in §4.4.2, my attributes are computed by setting the retention period to 1 hour, which is relatively expensive to maintain states, particularly at scale. Therefore, I investigate the impact of shorter retention period on the accuracy of my models. Table 4.6 shows various settings and their corresponding impact on both fine-grained and coarse-grained models.

In addition to model accuracy, I compute the average number of entries (*i.e.,*

Table 4.6: Size of data structure and accuracy of models as a function of retention period.

Retention period	**Avg. # entries**	**Fine-grained**	**Coarse-grained**
60 min	43.5M	98.88%	98.86%
30 min	32.4M	98.88%	98.86%
15 min	15.9M	98.91%	98.89%
5 min	5.4M	98.89%	98.82%
1 min	**959K**	**98.38%**	**98.53%**
30 sec	433K	86.84%	87.19%
15 sec	254K	72.58%	60.35%

key-value pairs) in the data structure for maintaining graphs in my dataset. It can be seen that both the model accuracy and the average number of entries (size of data structure) fall as the retention period gets shorter. The retention period of one minute seems to be the sweet spot in terms of accuracy (more than 98%) and size of data structure (less than a million entries), apparently, the one-minute period carries enough but not redundant information that reveal distinguishable patterns of each host type. For the rest of this chapter, I set the retention period to a minute.

Testing my Models on a Fresh Open Set

To evaluate the efficacy of my classification scheme, I test its performance against an open set derived from my 1-hour PCAP trace (which contains more than 2B packets from 21K enterprise hosts as discussed in §4.3) not shown to the model during the training phase. Attributes of hosts (with my ground-truth label) for both granularities are selected and calculated in the same way discussed above (*i.e.*, 1-min retention period with low-cost and medium-cost attributes of outbound traffic only).

Performance of the two models: Both models give high accuracy of 99.76% (fine-grained) and 98.57% (coarse-grained), interestingly slightly higher than those obtained during the cross-validation phase (in §4.4.3). Note that my open set (testing data) was obtained from the traffic of a busy working hour (*i.e.,* 9-10 am) when enterprise hosts are likely to be highly active and display clear network behavioral patterns. In contrast, the closed set (training and cross-validation data) corresponds to host behaviors during the entire day (covering both working and off-work hours).

Mis-classified instances: For those instances that were misclassified, the models display low confidence levels below 0.70. Manually investigating into their packet traces, I found two reasons for misclassification: (i) mixed functionality: a website proxy configured by a school also provides DNS resolution, file storage, remote accessing, and Redis proxy services, which are not recommended best practices for asset management and network security; and (ii) compromised and targeted by attacks: an end-host was correctly classified with high confidence levels (*i.e.,* ≥ 0.9) consistently for 13 minutes, and thereafter its predicted class with relatively low confidence levels (between 0.4 and 0.6) fluctuates between UDP server and UDP proxy for 47 minutes. I inspected its traffic during the low-confidence period and found that this host, in addition to its regular activities, was sending repeated DNS queries (asking for "`10.129.14.2xy.in-addr.arpa`")[8] at a constant rate of 2 packets-per-second to a public recursive resolver managed by ARIN. The host behaved like a bot-infected device in a query flooding attack [5] (possibly distributed across many bot devices).

Therefore, to help network administrators better identify cyber risks associated with their "suspicious" hosts (displaying unexpected behaviors and receiving low confidence scores from trained models), later in §4.5.1, I introduce a reactive mechanism using programmable networks that dynamically and selectively collects packets specific to the "focused" hosts for deeper investigation.

[8]I have partially obfuscated the IP address.

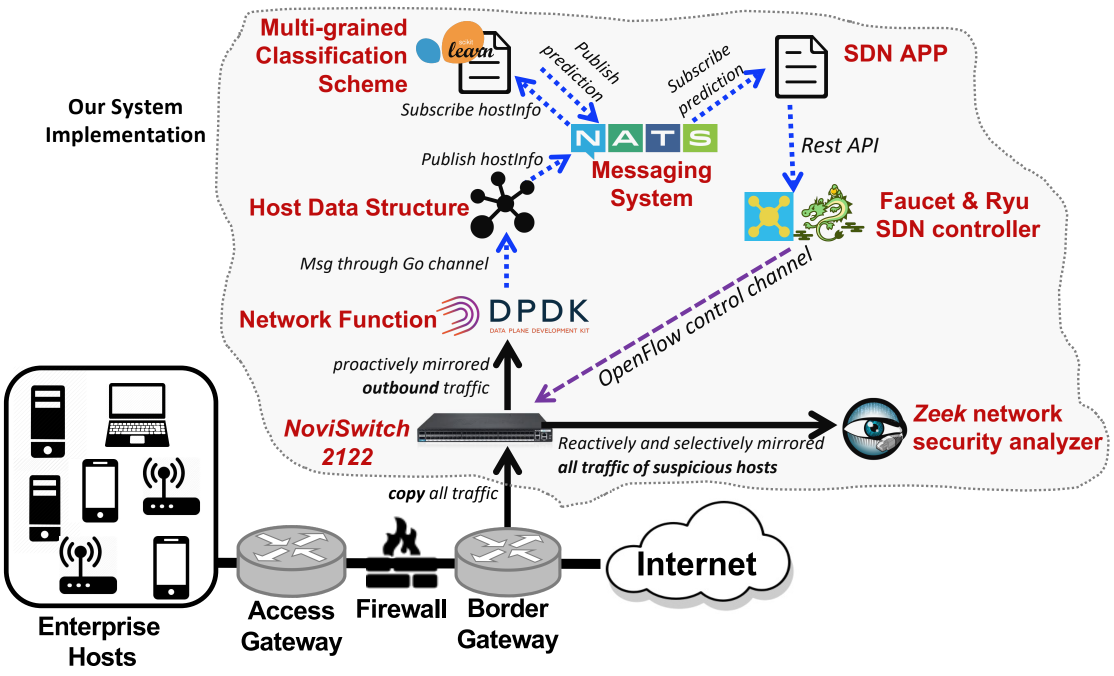


Figure 4.9: Prototype implementation.

4.5 Prototype Design, Implementation and Campus Field Trail

In this section, I prototype a practical system to: (a) classify enterprise hosts in real-time via my multi-grained classification scheme, and (b) dynamically isolate and inspect full traffic of "suspicious" hosts that are with low-confident predictions for a deeper packet-level diagnosis. Dynamic and reactive diagnosis is enabled by software-defined networking (SDN) techniques. I begin with the design of my prototype. I next draw insights into results of host classification and forensic analysis on the isolated traffic of suspicious hosts. Finally, I explain the system performance from a one-month trial (between 18th November and 18th December 2019).

4.5.1 Design and Implementation of my Prototype

I implemented my system using commodity hardware and open-source tools/libraries, and deployed it at the edge of my university campus network. Implementation details are shown in Fig. 4.9. My system receives inbound and outbound live traffic of the entire campus network (provisioned by the IT department of my university) via two 10 Gbps links. For my deployment, each functional block runs as an independent micro-service [239], and hence my system is more resilient to the failure of individual modules.

An SDN switch (NoviFlow 2122 [240]) is instructed by two separate SDN applications I developed for the Faucet SDN controller (inserting only proactive static rules) and the Ryu SDN controller (managing only run-time reactive rules). The two SDN controllers are logical independent so that the failure of one application (*e.g.,* inserting reactive rules) would not affect the other one (*e.g.,* inserting proactive rules). Proactive rules mirror outbound traffic to a generic server configured with Ubuntu version 16.04.4, which hosts my virtual network function (VNF) written in Golang using DPDK (data plane development kit) framework and the NFF-Go [241] library. The VNF parses packets, extracts required metadata, and updates my host data structure. Attributes of each host are calculated and periodically forwarded to the trained models (multi-grained classification scheme developed in Python3) via a messaging system (NATS) that acts as a data broker for information exchange. Predictions (classified labels and confidence levels) are published to the messaging system.

The hosts that receive a low confidence score (*i.e.,* less than 0.8) from the coarse-grained model (suspicious hosts) will need further and deeper investigation to determine the cause of deviation from their expected behavior. Such deep inspections for a host last till it receives a high confidence score. To achieve this, the SDN component of my system dynamically inserts reactive rules to mirror both inbound

Table 4.7: Summary of hosts with fine-grained types from the one-month campus field trial.

Fine-grained host type	**# host**	**avg consistency**	**avg consistency (wrk)**	**avg utilization**	**# previously unknown hosts**
Website server	362	0.93	0.98	0.33	**306 (85%)**
Authoritative name server	17	0.96	0.99	0.65	**2 (12%)**
VPN Server	13	0.94	0.98	0.41	**0** (0%)
Remote computing platform	159	0.83	0.89	0.34	**147 (93%)**
File storage server	18	0.93	0.95	0.26	**4 (22%)**
Mail server	19	0.98	0.92	0.34	**1 (5%)**
DNS proxy	9	0.81	0.88	0.37	**2 (22%)**
Web proxy	7	0.76	0.79	0.44	**3 (43%)**
NAT gateway	272	0.61	0.89	0.61	**13 (5%)**
End-host	18,891	0.99	0.97	0.17	**18,128 (96%)**

and outbound traffic of those suspicious hosts – rules are initiated by the SDN app to the Ryu controller through RESTful APIs.

To demonstrate the selective packet inspection of suspicious hosts, I set up Zeek [242] (a popular open-source security analyzer previously known as Bro). This software-based deep packet analyzer does not scale to a large volume of traffic. My system, instead, feeds Zeek with a minor fraction of the entire traffic, only that belongs to the hosts that are flagged suspicious by the inference models.

4.5.2 Insights into Campus Host Types

I now draw insights into the types of campus hosts predicted by my trained models and the consistency of their prediction by analyzing results of my system during the trial.

Fine-Grained and Coarse-Grained Classes

By analyzing the prediction results of my system during the field trial, I obtained the label (of network roles and behaviors) for a large set of hosts not managed by my organizational IT department. Table 4.7 and 4.8 summarize the classification results of fine-grained and coarse-grained types, respectively.

The hosts I determined their ground-truth type by DNS name (discussed in

Table 4.8: Summary of hosts with coarse-grained types obtained from the one-month campus field trial.

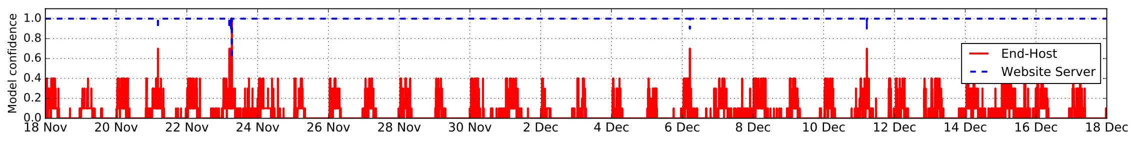
Coarse-grained host type	# host	avg consistency	avg consistency (wrk)	avg utilization	Low-conf. predictions	Low-conf. hosts
TCP-dominant Server	2,005	0.55	0.77	0.19	65,978	**231 (11%)**
UDP-dominant Server	144	0.79	0.81	0.25	23,061	**81 (56%)**
TCP-dominant Proxy	0	0	0	0	0	**0 (0%)**
UDP-dominant Proxy	8	0.17	0.81	0.38	2,919	**8 (100%)**
Non-typical NAT Gw.	19	0.43	0.85	0.56	971	**18 (94%)**
Non-typical End-host	1,946	0.88	0.92	0.24	230,029	**486 (24%)**

§4.3.2) have been cross-checked and validated. Note that some of those hosts (5 website servers, 4 remote computing platforms, and 1198 end-hosts) were not present on the network during the field trial – data used in §4.3.2 was collected nine months prior to the field trial. In addition, a total of 18,619 previously unknown hosts are classified to their fine-grained types for at least half of their active life during working hours (*i.e.,* 9am – 5pm), as shown by the rightmost column of Table 4.7, including 306 website servers, two authoritative name servers, 147 remote computing platform, four file storage servers, one mail server, two DNS proxies, three web proxies, 13 NAT gateways, and 18,128 end-hosts.

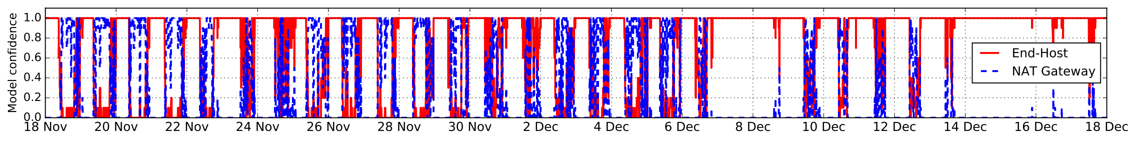
As expected, the fine-grained type can be determined confidently for a portion (*i.e.,* not all) of active hosts. The hosts, which do not receive a confident prediction from the fine-grained models, are classified by the coarse-grained model with relatively high confidence (above 0.80). I have found 1,774 TCP-dominant servers, 63 UDP-dominant servers, a non-typical NAT gateway, and 1,460 non-typical end-hosts. The coarse-grained predictions are accompanied by their dominant transport-layer services. For example, two TCP-dominant servers consistently used TCP/3306 (used by MySQL servers) as their top services, and one UDP-dominant server that is heavily active on UDP/427, suggesting a CIM server for managing hardware health information.

Behavioral Consistency of Enterprise Hosts

I note that some hosts may display different behavioral patterns during certain periods. Main servers (*e.g.,* organizational servers configured by the IT department) are



(a) Main website server: it is either classified as website server or end-host.



(b) NAT gateway: it is either classified as NAT gateway or end-host.

Figure 4.10: Time-trace of model confidence per class for two host examples: (a) website server and (b) NAT gateway.

consistently found to behave as their expected class with high confidence. However, subsidiary servers (managed by sub-departments) displayed variable behaviors during different hours. As an illustrative example, I show in Fig. 4.10(a) the time-trace of the model confidence for a server (the student web portal of my university), which is mostly classified as a website server with a few instances misclassified as end-host throughout my field trial. It can be seen in Fig. 4.10(a) that the model confidence is fairly high (close to 1) when this host is classified as a website server (dashed blue lines). I observe that the model confidence for predicting it as an end-host (solid red lines) is fairly low (mostly <0.4), with a few instances crossing 0.6 – only an instance exceeds 0.8, resulting in misclassification. I manually verified that this server undertakes routine maintenance (*i.e.*, fetching updates from the Internet) around midnight.

Proxies and NAT gateways tend to display varying profiles as their network activities depend highly on internal user behaviors. Therefore, these networked assets will likely get predicted as end-hosts during off-peak hours. For example, Fig. 4.10(b) illustrates the model prediction for a NAT gateway in my field trial. During working hours (9am – 7pm) on weekdays and weekends between 18th Nov and 7th Dec, the host was classified as a NAT gateway with high confidence (the blue dashed line). In contrast, it gets labeled as end-host with high confidence (the solid red line) during idle hours, including night time and days of study period (before final examinations of the academic term).

Note that each host may receive different predictions (classes) from the trained models during its lifetime. Therefore, the "consistency" of my inference models per enterprise host is an important metric to measure. I compute a measure of consistency (per host), which is the fraction of time the host is classified as its most frequent type (class). For illustration purpose, within a particular class (fine-grained and coarse-grained) I compute the overall average consistency (third column) and the average consistency during working hours 9am – 5pm that are respectively reported in the third and fourth columns of Tables 4.7 and 4.8. The average utilization (active fraction of lifetime) per class is shown in the fifth column of the two tables.

End-hosts and servers display a fairly consistent behavior compared to other types. Proxies and NAT gateways may behave as end-hosts during the inactive time, resulting in relatively low behavioral consistency. Hosts across all types (fine-grained and coarse-grained) display a more distinct behavior of their type (hence receive consistent prediction) during working hours.

Another observation is that many servers (especially those configured by sub-departments) are fairly under-utilized (the probability of being active is less than 30%). For example, four of 9 DNS proxies are adequately utilized (more than 16 hours a day), while others are relatively idle (*i.e.,* less than 2 hours of activity per day); thus can be candidates for getting merged (for economic and security management reasons) with other proxies on the network.

Lastly, for those hosts whose reliable prediction is only available by the coarse-grained model, 824 of them receive a low confidence score (less than 0.8) for a total of 322K prediction instances (details are shown in Table 4.8). Once a host receives a low-confidence prediction from the coarse-grained model, its entire traffic (inbound and outbound) is mirrored for a deep packet inspection – I use Zeek in my prototype.

The host remains under the deep-inspection mode until it receives a high-confidence prediction.

4.5.3 Insights into Suspicious Hosts from Deep Packet Inspection

During the field trial, the Zeek packet inspector raised a total of 381,499 packet-level alarms for 714 suspicious hosts, including 465 end-hosts, 18 NAT gateways, 6 UDP proxies, 45 UDP servers, and 159 TCP servers. Resulted alerts are from 32 types. The top alert types are "`truncated_tcp_payload`" indicating TCP-based attacks using crafted packets, "`possible_split_routing`" indicating single directional flows that may belong to scans and DDoS floods, "`data_before_established`" for potential volumetric anomalies, and "`inappropriate_FIN`" for TCP-FIN based anomalies.

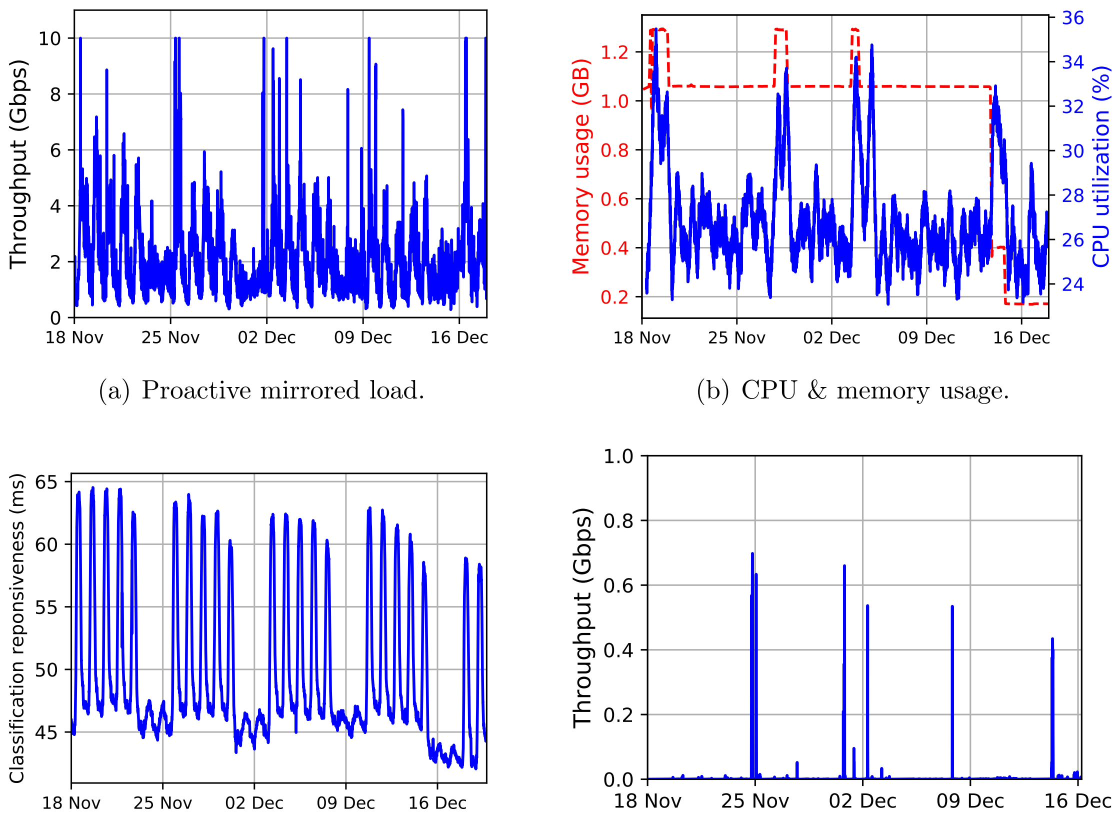
The distribution of alerts per host would help IT departments infer the root cause of their abnormal behaviors. For example, I found that a third of all resulted alerts correspond to only 11 suspicious hosts, suggesting extra attention in the forensics analysis. In what follows, I discuss my manual investigations specific to suspicious hosts whose behaviors triggered 8% (UDP proxy) and 5% (TCP server) of alerts, respectively. I emphasize that a more systematic forensic analysis is beyond the scope of this thesis and is left for future studies.

Let us start with the suspicious DNS (UDP) proxy, configured by an affiliated organization of my university, that consistently displays typical behaviors of UDP proxies most of the time. However, every 5 to 10 minutes (periodically), this host is misclassified (with low confidence) as a NAT server during peak hours and as an end-host or even TCP proxy during off-peak hours. Analyzing the Zeek logs, I found this host generates many TCP SYN-ACK packets followed by empty ACK packets targeting a range of TCP ports on external victims (on Microsoft Azure cloud) for a minute and then goes idle for 5-10 minutes. Such behaviors result in a large number of alerts of "`truncated_tcp_payload`", "`SYN_with_data`", and "`TCP_seq_underflow_or_mismatch`" for malformed TCP packets and incomplete connections. This host seems to be infected by malware to participate in TCP ACK-based scans [243] or flooding [244] activities.

Moving to the suspicious TCP server that purely offers TCP/3274 was classified as TCP server for most (86%) of its active time. For other times, I found the prediction of this host fluctuates between end-host and TCP server with fairly low confidence scores (about 0.3). Analyzing the alerts generated by the Zeek tool, I see that in addition to its typical inbound traffic, the server sent many outbound TCP SYN packets (without any response) targeting TCP/443 on a wide range of external victims (from a block of /16 IP address dedicated to Amazon cloud services) – about 5 packets per victim. These suspsicious packets led to frequent alerts such as "`window_recision`" and "`TCP_ACK_underflow_or_mismatch`". This pattern suggests that the host is possibly involved in SYN reconnaissance attacks, probing the availability on HTTPS service on external hosts as a preliminary step before launching reflection-based DDoS attacks [245].

4.5.4 System Performance

I now report the real-time performance my system during my field trial. As shown in Fig. 4.11(a), the throughput of the entire outbound network traffic (proactively mirrored) processed by the Network Function engine (which only processes packet headers). The rate of analyzed traffic varies between 0.3 Gbps to 10 Gbps, where daily peaks occur around mid-day on weekdays. In Fig. 4.11(b), CPU utilization of my server (which hosts traffic parsing VNF, host graph data structure, classification scheme, and SDN controller) follows a periodic pattern and is bounded between 24% and 36%, as shown by solid blue lines. Also, the memory usage varies from 0.2 GB to 1.3 GB, as shown by dashed red lines – note that the host-based data structure



(a) Proactive mirrored load.

(b) CPU & memory usage.

(c) Responsiveness.

(d) Reactive mirrored load.

Figure 4.11: Real-time performance of my system: (a) proactive mirrored load, (b) CPU and memory usage of the server hosting VNF, host data structure, inference engine and SDN controller, (c) responsiveness of the classifier (per epoch), and (d) reactive mirrored load.

contributes to majority of memory usage. Fig. 4.11(c) illustrates the responsiveness of the classification scheme called every minute, which is less than 65 ms even during peak hours, proving that my system can give real-time inference of enterprise hosts behavior. Lastly, reactively mirrored traffic load (to fine-grained packet inspection engine via my SDN-based mechanism) for "focused" enterprise IP addresses is shown in Fig. 4.11(d). It was light enough (*i.e.*, typically below 300 Mbps) to be processed by a computational extensive deep packet inspector.

4.6 Conclusion

Real-time classification of hosts and tracking their behavior are critical for enterprise network operators to manage their network assets effectively and securely. In this chapter, I developed an intelligent system that continuously classifies and monitors the network behavior of enterprise hosts. I conducted a large-scale analysis on traffic traces of an enterprise network and characterized network behavioral patterns of various host types at two levels of granularity. I then identified 256 attributes of host behavior and developed a multi-grained inference scheme consisting of a ten-class classifier and a six-class classifier that yields a high accuracy of 99%. Finally, I prototyped a practical system empowered by software-defined networking and virtual network functions and deploy it in a large university campus network. I presented insights obtained over a month field trial, such as the ability to identify hundreds of typical servers and their utilization and thousands of non-typical assets, and highlight anomalous behaviors pertinent to possible cyber-threats.

Chapter 5

Hierarchical Anomaly-based Detection of Distributed DNS Attacks on Enterprise Hosts

Contents

Domain Name System (DNS) is a critical service for enterprise operations, and is often made openly accessible across firewalls. Malicious actors use this fact to attack organizational DNS servers, or use them as reflectors to attack other victims. Further, attackers can operate with little resources, can hide behind open recursive resolvers, and can amplify their attack volume manifold. The rising frequency and effectiveness of DNS-based DDoS attacks make this a growing concern for organizations. Solutions available today, such as firewalls and intrusion detection systems, use combinations of blocklists of malicious sources and thresholds on DNS traffic volumes to detect and defend against volumetric attacks, which are not robust to attack sources that morph their identity or adapt their rates to evade detection.

I propose a method for detecting distributed DNS attacks that uses a hierarchical graph structure to track DNS traffic at three levels of host, subnet, and autonomous system (AS), combined with machine learning that identifies anomalous behaviors at various levels of the hierarchy. My method can detect distributed attacks even with low rates and stealthy patterns. My contributions are three-fold: (1) I analyze real DNS traffic over a week (nearly 400M packets) from the edges of two large enterprise networks to highlight various types of incoming DNS queries and the behavior of malicious entities generating query scans and floods; (2) I develop a hierarchical graph structure to monitor DNS activity, identify key attributes, and train/tune/evaluate anomaly detection models for various levels of the hierarchy, yielding more than 99% accuracy at each level; and (3) I apply my scheme to a month's worth of DNS data from the two enterprises and compare the results against blocklists and firewall logs to demonstrate its ability in detecting distributed attacks that might be missed by legacy methods while maintaining a decent real-time performance.

5.1 Introduction

The critical role of DNS infrastructure in large enterprises makes it a popular target for cyber-criminals. In recent years, distributed denial-of-service (DDoS) attacks based on DNS have risen in frequency, volume, and sophistication [246], and it is likely to worsen further as the attack surface expands with bring-your-own devices (BYOD) and Internet-of-Things (IoT) appliances [43, 247]. As an example, more than 100K compromised IoT devices were enslaved in 2016 for a global-scale DDoS attack on Dyn's DNS infrastructure [248] which prevented Internet users from accessing more than 1.2K web services such as Netflix, Spotify, and Twitter. According to EfficientIP [249], in 2020, nearly 79% enterprises suffered from DNS attacks which cost on average $924K per attack.

In spite of these growing risks, organizations today have huge DNS "blind spots" [170] that leave them exposed to DNS-based attacks. Large enterprises have many departments, each with their own information technology (IT) personnel, independently managing DNS servers/caches (this is particularly true in loosely-federated organizations like Universities), so it becomes very challenging to track and lock-down internal DNS infrastructure at a central firewall. Small businesses often rely on their Internet Service Provider (ISP) for security, and the ISP is quite likely to allow all DNS traffic through as they may not have visibility of the DNS infrastructure of the business, which can be dynamic. Further, even for organizations that do try to restrict incoming DNS traffic, malicious entities can spoof well-crafted DNS queries to reach and exploit their internal DNS services. New methods are therefore needed that are robust to dynamic DNS infrastructures and morphing attacks.

DNS-based attacks are broadly categorized into three groups: (1) query floods (also known as DNS flooding), mainly sourced from botnets that directly bombard victim servers (mainly authoritative name servers or recursive resolvers) with a large number of queries to exhaust the victim's resources, (2) DNS reflection/amplification attacks that utilize open DNS resolvers as proxies [250] by sending spoofed source address queries (using the intended victim's IP address), and (3) DNS scans [251, 252] that actively probe a target network to identify potential victims for future DNS floods and/or reflections. As emphasized earlier, current intrusion detection/prevention and firewall systems rely on static configurations, and are not robust enough to detect and block these attacks in the presence of dynamic DNS infrastructures and morphing attacks.

Most security solutions, both software-based tools and hardware-based appliances, typically use static *signatures* of known attacks. However, signature-based detection approaches are difficult to scale cost-effectively and require regular updates since attack vectors evolve rapidly [113]. Existing commercial intrusion detection systems often use threshold methods [202–204, 253] whereby they search for recurring patterns in traffic by counting the number of certain events occurring within a "defined period", and take action if the configured "threshold values" are exceeded.

Existing methods require IT departments to configure static policies and set threshold values for counting periods. Determining suitable thresholds is a non-trivial task since it is not obvious what the optimal values for effective defensive policies should be applied – high thresholds allow attacks to go undetected while small thresholds can result in a large number of false positives, incurring a high cost of investigation. If thresholds are specifically configured by administrators (*e.g.,* 120% of empirical peak load for a critical server as suggested by firewall vendors [202]), then existing firewalls can protect specific servers (or IP subnets) from being overwhelmed by an excessive rate of inbound packets affecting their normal service operation. However, they are still unable to identify and defend against distributed attacks (*i.e.,* identifying attack sources), since each source generates malicious traffic at low rates [26] (and may appear legitimate). I note that low-rate attacks (*e.g.,* CPU-exhaustion DoS attacks [254]) are still important to detect since they may be powerful enough to adversely affect servers with less resources. Also, distributed attacks often begin at a low rate before causing serious disruption or damage [255]. They often originate from bot devices under a subnet or autonomous system (AS) [256] that can only be detected by maintaining information for external entities at multiple levels of aggregation.

To address these shortcomings, I develop, implement, and evaluate an anomaly-based detection system, incorporating a dynamic and hierarchical graph structure of well-selected attributes, to capture real-time volumetric behavior of external hosts and detect external anomalous entities at various levels of aggregation. There exist prior works which detect attacks at destination networks, but their primary focus is to identify "victims". To the best of my knowledge, this work the first to propose a victim-side method for isolating " attack sources" in distributed DNS attacks (*i.e.,* DDoS floods and reconnaissance scans) depending on their nature of distribution across hosts, subnets, or AS.

My key contributions are summarized as follows. **Firstly**, in §5.3, I highlight the characteristics of malicious Internet hosts that launch DNS volumteric scans and flooding attacks on enterprise networks by analyzing datasets of real DNS traffic, consisting of approximately 400 million DNS queries and responses, collected from two enterprises over a week. **Secondly**, in §5.4, I develop a hierarchical graph structure with dynamic nodes and edges for monitoring the DNS query behavior of external entities at various levels of aggregation (namely host-, subnet-, and AS-level), identify attributes that can be computed cost-effectively in real-time, generate anomaly detection models using benign traffic only, and evaluate them using benign traffic as well as synthetically generated attack traffic. **Finally**, in §5.5, I demonstrate the efficacy of my scheme especially in detecting low-rate DNS-based attacks by replaying a month's worth of DNS traffic to my prototype, and validate my results by checking flagged external hosts against public blocklists of malicious IP addresses, as well as against logs from a commercial firewall.

5.2 Related Work

In this section, I survey related literature for the scopes my work falls in (*i.e.,* DNS security and defense of distributed network attacks) and highlight the research gaps that are addressed in this chapter.

5.2.1 DNS Security

DNS security has been an attractive topic for both industry and academia, especially on integrity of DNS records [179, 257, 258], and vulnerabilities of DNS infrastructure to volumetric attacks [2, 245, 259].

As for the integrity of domain names, malicious users on the Internet exploit DNS protocol to signal and control malicious network infrastructures such as DDoS botnet. Authors of [260] analyzed DNS responses to detect unusual behaviors (anomaly detection) related to domain names such as typo squatter domains and fast flux domains. *Kopis* [261] can accurately detect malware-related domains by using statistical features (such as distribution of requesters) at top-level domain servers. In [172], authors point out that attackers use domain generation algorithm to bypass detection systems, and they come up with a detection approach using clustering and classification algorithms for domain names and their requesters. Furthermore, as discussed in [174], malicious entities are able to launch resource-exhaustion attacks on DNS infrastructures by using disposable domains. Integrity problems can also arise during DNS lookups from legitimate users. Unsecured DNS communication can be easily hijacked and manipulated by third parties [257]. To address this problem, secured extensions such as DNSSEC [262] and DNS-over-HTTPS [193] have been proposed. However, Chung et al. revealed that the adoption of DNSSEC is still in early stages [182].

Researchers have also investigated vulnerabilities of DNS services and infrastructure to volumetric attacks including DDoS and reconnaissance scans. DNSSEC is seen more attractive by DNS amplification attackers. Rijswijk-Deij et al. [185] showed that DNSSEC can be mis-used for larger amplifications (in reflection attacks) compared to standard DNS. Work in [259] actively probed DNS resolvers available on the Internet and quantified their reflective capabilities.

In this work, I develop data-driven models, trained by real enterprise data, to detect and identify external anomalous sources (at three hierarchical levels including host, subnet, and AS) that attempt to discover, attack, or exploit enterprise DNS infrastructure, even when they reduce their activity profile by getting distributed and reducing their traffic rates.

5.2.2 Defense of Distributed Network Attacks

Efficient defense of distributed network attacks have been widely-studied by many researchers. Works can be classified based on where they are deployed regarding the anatomy of attacks, *i.e.,* at-source, at-destination, or at-network [78]. At-source methods are deployed at the place where attacks originated, *D-WARD* [80] looks for suspicious traffic patterns (*e.g.,* source IP spoofing on incoming traffic) and applies rate-limiting to corresponding hosts at the network edge. *ShadowNet* [263] measure traffic attributes (*e.g.,* rate of HTTP GET requests) from edge routers serving IoT devices to detect IoT botnets. For at-network solutions, *LADS* [264] uses lightweight SNMP and NetFlow statistics collected from an ISP's backbone routers, and *Bohatei* [11] uses SDN to reroute suspicious traffic (*e.g.,* excessive number of TCP SYN packets) to IDS middleboxes on ISP networks. *PSI* [232] is an example of at-destination approach. It is deployed at the edge of an enterprise network which dynamically applies security rules of firewalls or IDS to traffic of interest (*e.g.,* certain protocols) for detecting attacks towards enterprise hosts with optimized accuracy and cost. With emerging SDNs, denial-of-service attacks on controllers start ramping up. *SWGuard* [265] defenses against control-plane reflection attacks by monitoring victim's down-link control messages.

Note that both at-source and at-network systems are effective in both detection and mitigation but they require automatic coordination (signaling) between network operators on the path between source and destination, while legacy at-destination methods can well isolate victims but not source of attacks. My work is the first to develop a novel hierarchical graph structure for detecting attacks at-destination that can precisely identify external attack sources at IP-level to subnet and AS levels, even the sophisticated distributed attackers which keep their individual attack traffic rates so low, bypassing network firewalls. Comparing my work with systems against volumetric distributed attacks in particular, existing methods primarily aim to detect potential victim servers (instead of identifying attack sources) [10, 232, 266, 267]. Moreover, their objective upon detection of an attack is to either isolate or rate-limit the victim, affecting malicious and benign traffic sources alike. My work, instead, aims to detect distributed attacks and identify their sources which can be precisely blocked (automatically or manually) for remedial action without affecting benign/legitimate sources.

5.2.3 Summary of Research Gaps

I have identified three research gaps in relevant existing works. First, no prior work systematically characterized the behavior of malicious entities that generate distributed DNS attacks on enterprise networks. In §5.3, I highlight the anatomy of distributed DNS attacks on enterprise assets. Second, no prior solution considered the context of attack sources in terms of their subnets and ASes. I, instead, develop a hierarchical data structure (§5.4) to track the behavior of attackers at various aggregate levels. Third, existing enterprise-side defense systems predominately aim to detect victim internal servers which are under attack (*i.e.,* identifying "destination" of attacks), and hence do not distinguish between malicious and benign external sources (*i.e.,* unable to identify "source" of attacks, especially when they are distributed). I, instead, detect distributed attack sources (§5.5) even those which keep their traffic rates relatively low to bypass security appliances.

5.3 Profile of DNS Volumetric Queries to Enterprise Networks

In this section, I analyze the DNS traffic collected from the border of two enterprise networks, a large University campus and a medium-size research institute to profile incoming queries to enterprise hosts. My data analysis primarily focuses on the behavior of external source entities (*i.e.,* hosts, subnets, and ASes outside the protected enterprise network) that may contact or attack DNS servers in any enterprise settings. Note that the two studied networks have rich DNS facilities (*i.e.,* authoritative DNS servers, recursive resolvers and public-facing servers with assigned domain names) which are frequently targeted by distributed DNS attacks. Smaller organizations with fewer DNS servers can become target of similar inbound attacks, perhaps at relatively lower frequency.

In both instances the IT department of the enterprise provisioned a full mirror (both inbound and outbound) of their Internet traffic (each on a 10 Gbps interface) to my data collection system from their border routers (**outside** of the firewall),

Table 5.1: Summary of dataset during 1-7 May 2018.

	University Campus	**Research Institute**
Query In	139,136,237	99,847,262
Response Out	97,008,082	61,164,416
Unanswered Query	42,127,345	35,332,068
Invalid Query	5,335,019	9,335,751
`NameError`	4,935,385	4,154,238
`ServerFailure`	264,100	490,045
`Refused`	135,404	4,640,829

and I obtained appropriate ethics clearances[1] for this study. I extracted all DNS packets from each of the enterprise Internet traffic streams in real-time by configuring OpenFlow match rules for incoming/outgoing IPv4 port 53 packets on an SDN switch. My dataset was collected during the full month May 2018, though my analysis in this section focuses on a one week period (1-7 May 2018), consisting of 139.1M and 99.8M incoming DNS queries to, along with 97M and 61.2M outgoing DNS responses from, hosts of the university and research networks respectively – a brief summary of the dataset for the first week of May 2018 is shown in Table 5.1. As an additional note, the work described in this chapter is done prior to the work in §3, thus, the dataset used in this chapter is collected earlier than that in §3.

5.3.1 Incoming DNS Queries

A benign incoming DNS query to an enterprise network typically targets an authoritative name server and the corresponding server responds with a `NoError` flag. There are, however, other types of incoming queries observed in real networks: **unanswered queries** (*i.e.,* DNS queries with no response) and **invalid queries**, those answered with flags other than `NoError`, for various reasons such as the query packet is malformed or corrupted, the query name does not exist or is not relevant for

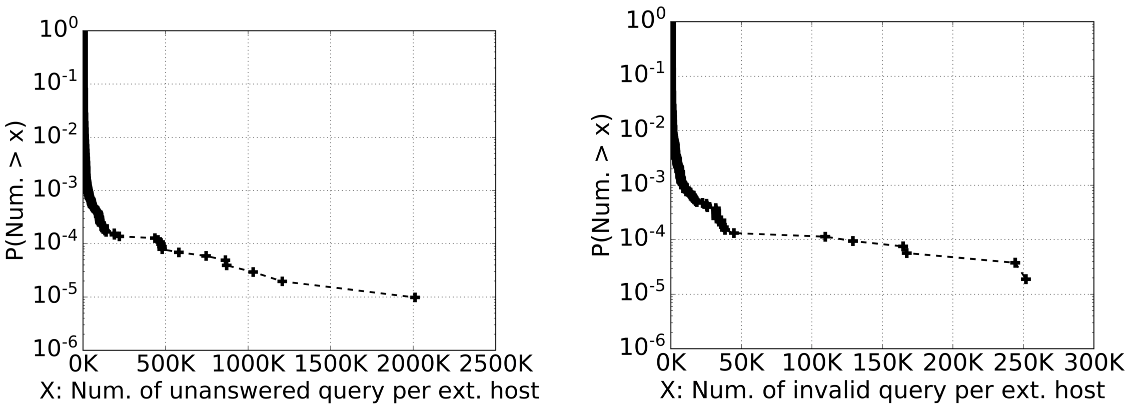
[1]UNSW Human Research Ethics Advisory Panel approval number HC17499, and CSIRO Data61 Ethics approval number 115/17.

the organization (and thus does not resolve to any IP address), or an unintended recursive resolver is queried.

Unanswered queries: During the first week of May 2018, I found a total of 42M and 35M unanswered incoming queries to the networks of the university and the research institute respectively – approximately one third of the total incoming queries in both organizations. I have verified that these queries typically targeted non-DNS servers inside the organization (by performing reverse DNS lookups) or IP addresses that are not active/present in the network to receive the query, and therefore were not answered.

Invalid queries: I also found a total of 5.3M and 9.3M invalid incoming queries to the networks of the university and the research institute respectively – accounting for 4% and 9% of the total incoming queries. For the university network, 92.5% of invalid queries were answered with a `NameError` flag, indicating that the DNS server was not able to resolve the queried name. Also, 5.3% of responses had an error code of `ServerFailure` and 2.5% an error code of `Refused`. Additionally, a tiny fraction of responses had `NotImplemented` and `FormatError` flags set (118 and 12 invalid queries respectively) – these error codes indicate incorrect messages contained in the queries or incorrect destination name servers (*e.g.,* a wrong domain name for the authoritative name server of the organization).

For the research institute, the distribution of error codes was as follows: 44.4% with `NameError`, 5.2% with `ServerFailure`, 49.6% with `Refused`, 0.4% with `FormatError` and only 117 instances of `NotImplemented` with additional codes including `NotAuth` and `NXRRSet` seen in 31,540 and 24 of responses to invalid queries. The last two error codes correspond to wrong destinations or messages in the queries.



(a) Unanswered queries. (b) Invalid queries.

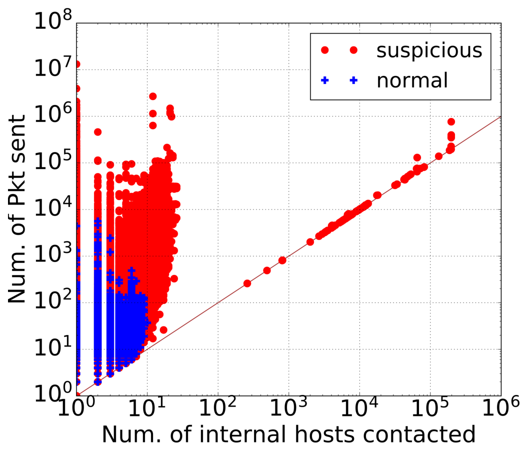
Figure 5.1: CCDF of: (a) unanswered, and (b) invalid, query counts per external host for a duration of one week (to university network).

5.3.2 External Hosts Sending Unwanted DNS Queries

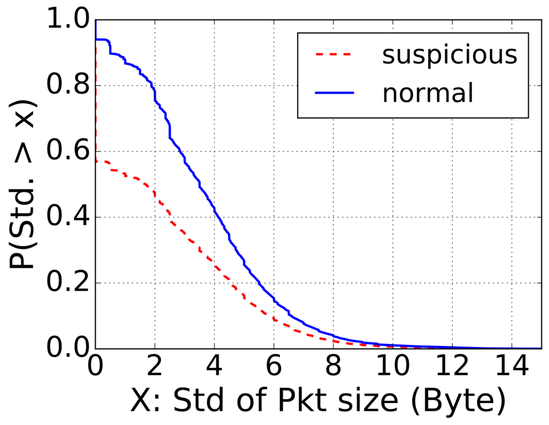
Considering all external hosts sending DNS queries to the university network[2], I counted a total of 168,538 unique hosts (*i.e.,* IP addresses) in my dataset for the first week of May 2018. These external hosts come from 46,729 distinct subnets and 16,775 unique autonomous systems. I found a total of 112,704 external hosts sent some form of unwanted query: 41,893 external hosts sent both unanswered and invalid queries; 59,907 sent only unanswered queries and 10,904 sent only invalid queries.

Interestingly, I observe that only a tiny fraction of external hosts are very active in sending unwanted queries. In Fig. 5.1, I show CCDF plots of unwanted queries counts per external host. Note that there are 29 external hosts each generated more than 100K unanswered queries, as shown in Fig. 5.1(a), with three hosts each generating more than a million unanswered queries over the week. Similarly, only 6 external hosts each sent more than 100K invalid queries, as shown in Fig. 5.1(b). I will see later in this section that these heavy hosts are indeed involved in volumetric-based DNS attacks in the form of host scanning and/or query flooding.

[2]In this subsection, I have omitted analysis of the research institute data so as to concentrate on insights. Similar observations were made for both organizations.



(a) Pkt count vs queried hosts count.



(b) Variance of query packet size.

Figure 5.2: Query behavior of external hosts during a week: (a) number of packets sent versus number of internal hosts contacted per individual external host, and (b) CCDF of query packet size (Bytes) per external host – normal and suspicious external hosts are highlighted in blue and red respectively.

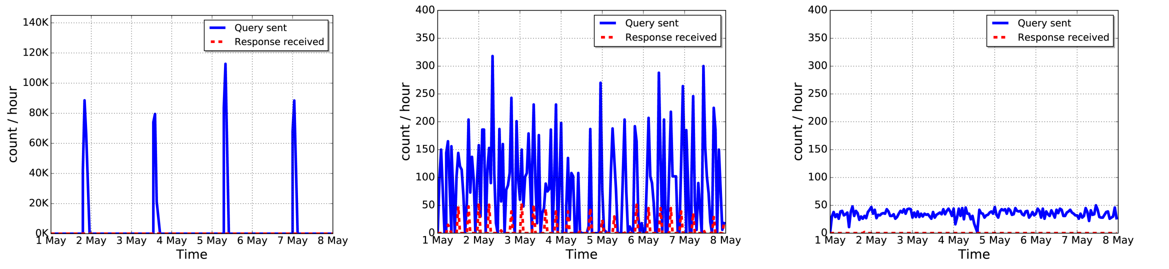
I now divide the external hosts into two groups: hosts with “normal” and hosts with “suspicious” behavior in their DNS queries – normal hosts do not send any unwanted DNS queries to the network. Note that some of the suspicious hosts may just have typographical errors in their DNS queries. Suspicious hosts are distributed across 30,295 subnets of 11,539 autonomous systems. I found that 73 ASes each with more than 100 hosts account for 59% of all suspicious IP addresses, of which the top 5 heavy ASes contain 19% of all suspicious hosts. This is not surprising as approximately half of all cyber-attacks originate from compromised devices in a small number of countries with insecure infrastructure [256]. On the other hand, 55,834 normal hosts are distributed across 10,706 unique ASes – the number of normal hosts for each AS is evenly distributed.

Fig. 5.2 illustrates the difference between the query behavior of normal and suspicious external hosts. Starting with the scatter plot of query packets count versus number of unique internal hosts queried for each external host, shown in Fig. 5.2(a), a cluster of normal hosts (shown by blue cross markers) is clearly visible at the bottom left corner of the plot – a normal external host does not send more than 10,000 queries in a week. On the other hand, suspicious hosts (shown by red circle markers) display two clusters (*i.e.,* suspicious hosts lying on the line $y = x$ that send one query to a large number of internal hosts, and suspicious hosts clustered on the left that send a large number of queries to a limited set of internal hosts).

Moving to the variance of the queries, I show in Fig. 5.2(b) the CCDF plot of the standard deviation of packet size sent by each of the normal and suspicious external hosts. It can be clearly seen that suspicious hosts (as shown by dashed red lines) display a smaller variation in the size of their query packets compared to normal hosts, highlighting a set of repopulated query names were automatically sent by suspicious hosts (*i.e.,* a bot). More importantly, I observe that a large fraction (*i.e.,* 42%) of suspicious hosts had identical packet size (*i.e.,* zero variation) for their queries during the week.

I also analyzed the payload of queries sent by each external host. Among suspicious hosts (excluding 4,436 hosts with just one query), I identified 19,551 hosts each used identical query names for all queries during the week, with many of these hosts having a large number of queries sent to the university network (*e.g.,* an external IP address sent 13,144,130 identical query packets to one internal IP address). I also observed that most of these suspicious query names are not relevant to the services provided by the enterprise. For example, one IP address (located in Russia) sent 763K queries with the query name "`com`" to the campus network, and another IP (located in Lithuania) sent 397K queries for "`nrc.gov`" during the week.

It is important to note that using identical query names is also seen for normal hosts. I note that 8,836 normal external hosts each queried only one domain name, but each of these hosts generated only a small number of queries during the week. Another observation for these specific normal hosts is that the DNS ID (*i.e.,* a 16-bit identifier in the DNS header) varies over time, whereas for suspicious hosts only one DNS ID was consistently used for successive DNS queries. Next, I analyze in more detail the properties of the two types of unwanted DNS queries.



(a) Periodic (an external host from Russia). (b) Focused (an external host from China). (c) Low-rate (an external host from US).

Figure 5.3: Weekly time-trace of various types of DNS scans: (a) periodic, (b) focused, and (c) low-rate; on enterprise hosts from the Internet.

5.3.3 DNS Query Scans

The first cluster of suspicious hosts shown in Fig. 5.2(a) corresponds to scanners that sent one query packet (*i.e.*, probe) to a large number of (potentially all) internal IP addresses. I note that the university owns three "`/16`" IPv4 address prefixes, which represents more than 196K unique IP addresses for internal hosts, as indicated by the largest number of hosts contacted in Fig. 5.2(a). Scans are typically performed by malicious entities to make a list of available DNS servers inside enterprises that could subsequently be used as reflectors for DDoS attacks [252]. Also, there are a number of "white hat" researchers who conduct DNS query probing to only detect (not attack) vulnerable DNS servers available on the Internet [35, 259, 268]. Scan queries are typically crafted packets with the query name field most likely not relevant for the enterprise network. If this query reaches an operational DNS server, a response may or may not be returned (depending on the particular name server configuration). Scanners can choose various strategies (in terms of the query rate) to perform reconnaissance tasks. Next, I consider three types of scans – a representative of each type is shown in Fig. 5.3.

Periodic Heavy Scans

Some scanners choose to scan the network on a periodic basis. Fig. 5.3(a) shows a scanner from Russia that conducted 4 large-scale scans during the week by sending queries to a total of 196,865 IP addresses (*i.e.,* almost the full three `/16` prefixes) inside the university campus. Each scan lasted for about 2 hours and the query name "`com`" was repeatedly used in all query packets. Forty one DNS servers (*i.e.,* recursive resolvers and authoritative name servers) inside the organization responded to this scanner – a majority of servers were not able to respond with a `NoError` flag to these queries: 24 servers responded with error code `Refused`, 7 with `ServerFailure`, and 2 with `NameError`. Surprisingly, 8 DNS resolvers (verified by reverse lookups) inside the enterprise network responded with `NoError` flag to the scanner, resolving the queried name "`com`". I note that the answers to this top-level domain name are fairly large, resulting in an attractive amplification factor (*i.e.,* ratio of response size to query size) of up to 43. I will see later in §5.3.4 that some of these servers were used as reflectors.

Focused Scans

Instead of blindly sweeping the entire IP range of a target network, some scanners (those with prior knowledge) may focus on selected IP addresses in their dictionary. Focused scans aim to validate the availability of potential DNS servers that can be used as reflectors/victims – some vulnerable DNS servers may subsequently be secured by a change of configuration or may no longer be operational.

A sample of focused scans is shown in Fig 5.3(b). This scanner (located in China) targeted 18 IP addresses from 2 subnets of the university network by periodically looping over a static list of hosts. Each scan round consisted of sending one customized query with the name "`d.c.b.a.in-addr.arpa`" to each IP address "`a.b.c.d`" – this reverse lookup query causes the victim (*i.e.,* potential DNS server) to return its own DNS name inside the organization. I observed that the same loop was repeated 760 times during the week. I verified (by manual reverse lookup) that 2 of these IP addresses are the university's main recursive resolvers, 3 of which are resolvers at Faculty/Department-level, and 13 IP addresses are WiFi gateways inside the enterprise network. Among these 18 internal hosts, only one department-level DNS resolver responded with the "`NoError`" flag, revealing its identity. Other DNS resolvers were securely configured and did not respond to this scanner – enterprise resolvers are meant to serve only internal hosts.

Slow-Rate and Distributed Scans

As heavy scans (*i.e.,* high rate queries) can be easily detected by current firewalls and intrusion detection systems, sophisticated scanners may choose to go under the radar by lowering their query rate or distributing the probing task across a number of hosts. Fig. 5.3(c) shows an example of a low-rate scan. The external host (located in US) consistently sent approximately 40 queries per hour with the question name "`VERSION.BIND`" – in total 5831 queries were sent to 5806 IP addresses during the week (*i.e.,* on average one query per internal host). I note that this query (specific to the most widely deployed DNS server application BIND) asks the server's version. I found that only one host (*i.e.,* a department-level DNS resolver) replied to this query with error code `Refused`. Interestingly, right after this response, the scanner sent another query with question name "`direct.shodan.io`" and the DNS resolver successfully responded with `NoError`.

Impatient low-rate scanners tend to distribute their task across multiple sources (more likely from a subnet/AS under their control). In my dataset from the university network, I found (via manual analysis) 6 distributed scans each originating from a distinct subnet. For each of these scans I observed a similar scan pattern (time-series, total count, query name) across all hosts involved. For example, in one of these distributed scans, 16 external hosts from a /22 prefix (located in US) each sent about 75K queries (one query per internal host contacted) asking for "`dnsscan.shadowserver.org`".

I also found a distributed scan sourced from multiple prefixes within a distinct AS – manually finding this type of distributed attack is a non-trivial task. In this scan, only three external hosts each from a different subnet (*i.e.,* one in /17, one in /12, and one in another /12 subnet) all associated with AS4134 (located in China), each generated about 130K queries to 65K internal hosts (*i.e.,* two queries per internal host) asking two domain names "`www.163.com`" and "`version.bind`".

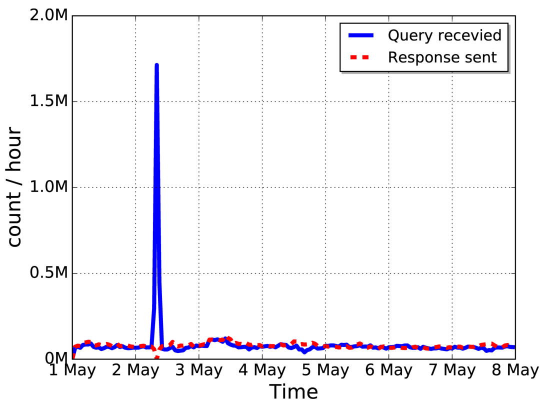
5.3.4 DNS Query Floods

I now consider the second cluster of hosts generating unwanted queries, shown in Fig. 5.2(a). These external hosts flood (sending a large number of queries to) a small number of enterprise hosts. I note that some flooders aim to exhaust the resources of the enterprise host (primarily DNS servers) [43, 269] whereas others aim to use enterprise servers to reflect/amplify volumetric DNS traffic to third party victims [184]. In the latter scenario, the attacker spoofs the source IP address of the query by using the intended victim's address.

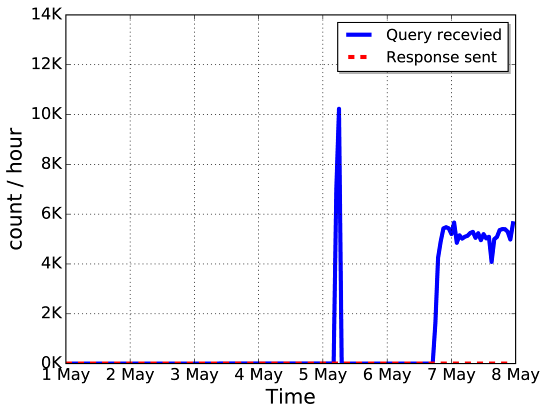
Similar to scanners (described in §5.3.3), flooders may use one or a list of query names which may or may not be relevant to the enterprise network. Due to the objective of flooders, I expect to see a higher rate of queries coming from external hosts to the enterprise network. Next, I analyze two types of DNS query floods with supporting examples from the two enterprise networks.

Flooding Enterprise Servers

DNS flooders may target a DNS server of an enterprise to exhaust its computational resources (by asking it to resolve an excessive number of queries) or a non-DNS



(a) Targetting an internal name server.



(b) Targetting an internal web server.

Figure 5.4: Weekly time-trace of DNS flooding on enterprise hosts from the Internet, targeting an internal (a) name server, and (b) web server.

server to consume its network resources. Fig. 5.4(a) shows the DNS traffic pattern (incoming queries and outgoing responses) for one of the main authoritative name servers inside the research institute. I can see that this server typically handled approximately 50K query packets per hour for domain names associated with the research organization – number of queries and responses are almost the same during normal operation (*i.e.,* except for the spike period of an attack on 2 May). However, between 06:48AM to 08:11AM on 2 May, this server received a surge of queries (*i.e.,* about 1.7M queries per hour) sourced from 29,614 external hosts, 4,053 of which kept sending repeated queries with the research organization domain name (the characters were randomly in capital or lower case, *e.g.,* "`reSeaRChInstituTe.OrG`" and "`ResEaRchINSTituTe.oRG`"). The highly suspicious external hosts were associated with 432 ASes (122 in U.S., 40 in Australia, 17 in Canada, and 17 in Brazil), where 26 ASes cover 3183 (78.5%) flooders. I note that the top two ASes (both in US) account for 1,211 and 510 flooders.

I can see that this large scale attack resulted in no query (legitimate or malicious) being responded to, as shown by red lines hitting zero during the attack period in Fig. 5.4(a). I note that this might be because either the server became non-operational, or the enterprise border firewall had detected the attack and possibly dropped all queries to protect the server.

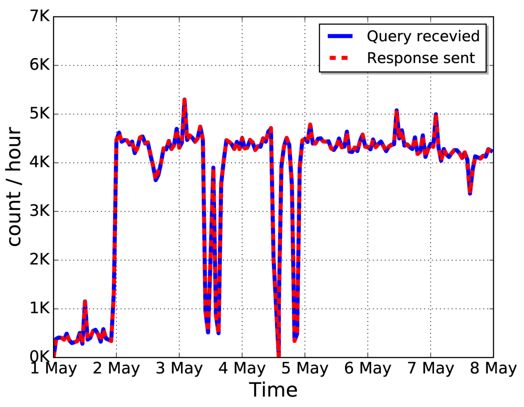
Fig. 5.4(b) shows a sample time-trace of DNS traffic for the second type of victims (*i.e.,* a non-DNS host). The victim host is a department-level web server in the university network. I can see that this web server does not typically receive DNS traffic but two instances of query floods hit this server: spikes on 5 May and 6-7 May. During 04:33 to 05:51 on 5th May, 18 external IPs each generated about 800 queries with the question name of either "`qjnntx.eleximg.com`" or "`static.mobike.com`". These suspicious external hosts are associated with 4 ASes, of which 2 heavy ASes (AS20473 and AS36351 located in US) account for 10 and 6 attackers – it is a common practice for attackers to develop their botnets within a compromised subnet or AS [256].

For the second instance, between 19:03 on 6th May till the end of the week, 11 flooders from 2 ASes (7 IP addresses from AS36351 in the US and 4 IP addresses from AS132203 in China) each sent about 15K queries to the web server with irrelevant query names from the list of "`qjnntx.eleximg.com`", "`c.afekv.com`", and "`global-ldns.v3.apsv1.com`".

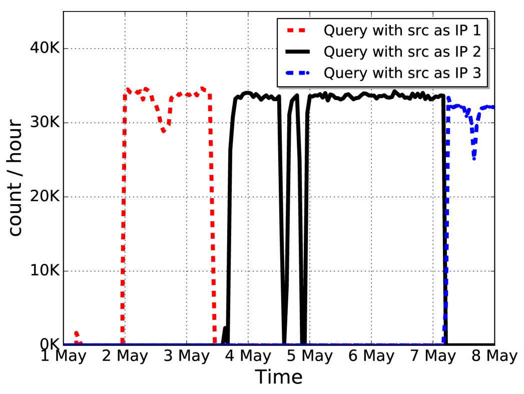
Reflective DNS Floods

As mentioned earlier, internal DNS servers of enterprises are targets for cyber-attackers for reflecting volumetric DNS traffic to third party victims on the Internet. Enterprise DNS servers are often discovered prior to this type of attack (as explained in §5.3.3). I note that the source IP address of queries in reflective DNS floods are spoofed using the intended victim's IP address, therefore external victims are perceived as external flooders by the border device of enterprise network.

I found one example of such an attack in my dataset for the university network that was well coordinated and persisted for almost a week by involving five internal DNS resolvers reflecting to three victims on the Internet, as shown in Fig. 5.5. 5 DNS resolvers (those that successfully responded to periodic scans) inside the university



(a) Behavior of an internal resolver.



(b) Aggregate queries from 3 victims to 5 internal DNS servers.

Figure 5.5: Weekly time-trace of a reflective DNS flooding attack: (a) one attack reflector (an internal DNS resolver), and (b) three external victims.

network simultaneously received a surge of DNS query packets (*i.e.*, around 4K-5K per hour) at around 11pm on 1st May with the question name "**`ietf.org`**". I note that the response size (in bytes) varies in the range of 15 to 45 times (*i.e.*, the amplification factor) the query size. Fig. 5.5(a) shows the query count (and the corresponding response count) for one of these DNS resolvers – others displayed almost the same pattern with a slight variation in their traffic rate.

Considering aggregated query traffic (with the victims' addresses as source) in Fig. 5.5(b), it is evident that the three victims were targeted consecutively (each shown by unique line color). I note that all of three victims are servers associated with AS49453 located in The Netherlands.

5.4 Volumetric Profiling and Detection Scheme

In this section, I present my methodology in profiling and detecting distributed DNS attacks by developing a dynamic volumetric behavior model, and employing anomaly detection algorithms. Fig. 5.6 illustrates the schema of my method. I first develop (§5.4.1) a binary-hierarchical attributed graph data structure to describe the

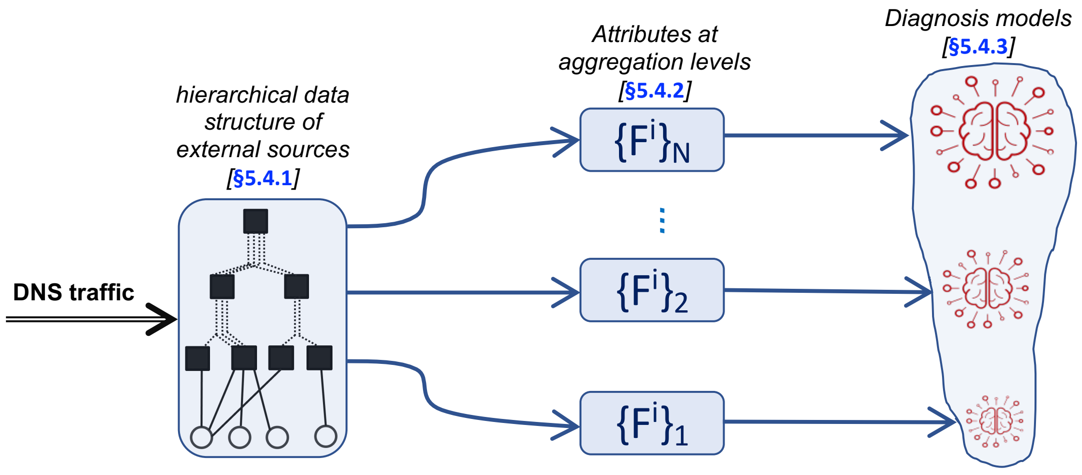

DNS traffic
hierarchical data structure of external sources [§5.4.1]
Attributes at aggregation levels [§5.4.2]
$\{F^i\}_N$
$\{F^i\}_2$
$\{F^i\}_1$
Diagnosis models [§5.4.3]

Figure 5.6: Architecture schema of my methodology.

volumetric traffic profile of external entities and mathematically show its efficacy in detecting attacks (especially distributed ones) by simple thresholding at multiple levels of the hierarchy. My data structure is applicable to generic volumetric scans and attacks. In this chapter, I only demonstrate its merits specifically to DNS-based attacks. I prove by mathematical analysis that my scheme is able to detect scans and floods of various forms (*e.g.,* distributed attacks are detected at an aggregated level) and varying rates (*e.g.,* low-rate attacks are detected within a guaranteed time period). Legacy threshold-based diagnosis methods only consider traffic rate as attribute for attack detection that makes it relatively easier for stealthy attackers to subvert the diagnosis systems. To address this shortcoming, I identify (§5.4.2) key attributes of network traffic that are collectively able to distinguish benign versus malicious behavior of external sources. Using these attributes, I extend (§5.4.3) my theoretical threshold-based hierarchy to a practical machine learning-based diagnosis system that employs anomaly detectors at three layers of host, subnet, and AS. I evaluate the performance of my scheme (profiling external sources using my novel data structure combined with ML-based models) in detecting distributed attacks with high accuracy, and highlight its superior detection ability in comparison with simple and hierarchical thresholding-based diagnosis methods.

5.4.1 Hierarchical Data Structure

In order to profile the volumetric behavior of an attacker, I employ a graph-like data structure to capture the relationship between external source entities and internal destination hosts of a given network. External source entities can be identified by a hierarchy of: hosts under subnets under ASes. As noted in §5.3, DNS query scanners and flooders are likely to be located within certain ASes and/or subnets. Therefore, it is important to develop a comprehensive model that covers traffic activity of external sources at various levels of aggregation (as opposed to purely individual hosts level), enabling us to detect sophisticated attacks that get distributed across a number of hosts, subnets, or ASes, aiming to evade threshold-based diagnosis by lowering their traffic rate.

Key Design Rationale

External entities can be identified by a hierarchy of: hosts under subnets under ASes. As noted in §5.3.2, DNS query scanners and flooders are likely to be located within certain ASes or subnets. Therefore, it is important to consider a comprehensive model that covers traffic activity of external entities at various levels of aggregation, enabling us to detect sophisticated attacks that are distributed or go under the radar (by lowering their traffic rate). Since the set of external entities can be quite massive (*i.e.,* potentially the whole IPv4 space on the Internet), it becomes impractical to keep states forever. On the other hand, forgetting states too quickly may result in missing slow attackers. Therefore, I need an efficient retention policy to age out inactive entities in my data structure.

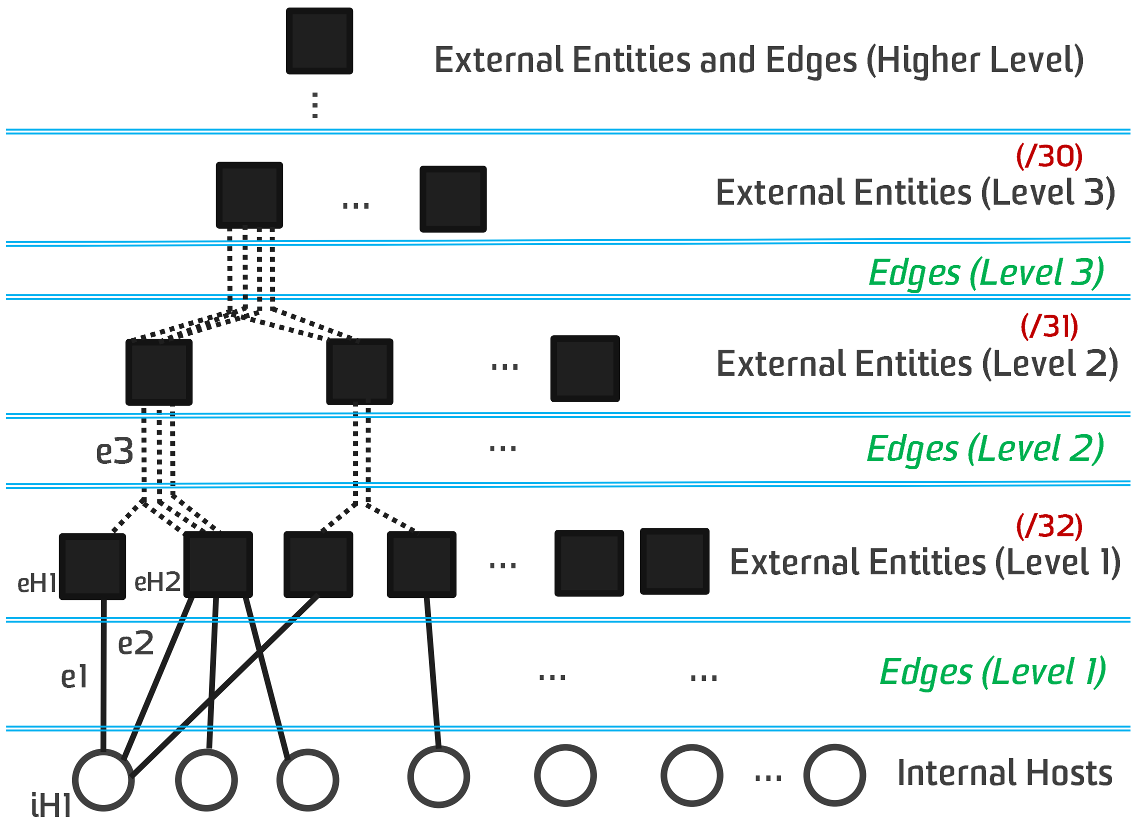

External Entities and Edges (Higher Level)
(/30)
External Entities (Level 3)
Edges (Level 3)
(/31)
External Entities (Level 2)
e3
Edges (Level 2)
(/32)
External Entities (Level 1)
eH1
eH2
e2
e1
Edges (Level 1)
iH1
Internal Hosts

Figure 5.7: My hierarchical data structure.

Theoretical Framework of My Data Structure

Given the above requirements, I construct a binary hierarchical attributed graph model [66], as shown in Fig. 5.7. In this diagram, enterprise internal hosts are represented by circles at the bottom, connected via solid edges to external entities (shown by filled squares) which themselves get aggregated to upper-level entities via dashed edges. I apply aggregation to both nodes and edges in this model. Level-1 entities each represents an external host IPv4 address on the Internet (*i.e.,* /32), while level-2 entities are a group of external hosts created by masking one bit in the IPv4 address (*i.e.,* /31) – all the remaining levels work by incrementally masking the IP to get a larger subnet. To visualize the edge aggregation let us focus on two leftmost level-1 hosts (*i.e.,* $eH1$ and $eH2$) shown in Fig. 5.7. They both have a connection (*i.e.,* $e1$ and $e2$) to the first internal host (*i.e.,* $iH1$). These two level-1 edges are aggregated as a single edge $e3$ at level 2.

Dynamic retention policy: Nodes (*i.e.,* an external entity) and edges in my model each would have a set of attributes (explained in §5.4.2), describing their profile, and they will be removed dynamically from the graph if they become inactive for longer than a corresponding retention duration defined in Algo. 1.

Algorithm 1 Multi-thresholding Retention Duration

```
1: procedure ActionPktIn
2:     while PktIn do
3:         for $n$ from $MaxLevel$ to 0 do
4:             if $PktIn$ not match any $node$ on level $n$ then
5:                 create corresponding $node$ and $edge$ with
                   $retentionDuration$ as $2^{n-1}R$
6:             else
7:                 update the matched $node$ and $edge$
8:             end if
9:         end for
10:    end while
11: end procedure
12: procedure ActionAttackerDetected
13:    for $n$ from $maxLevel$ to 0 do
14:        if $node_{n,i}$ on level $n$ is anomaly then
15:            if $n > 1$ then
16:                set retention duration of its child nodes
                   on level $n-1$ to their parent level
17:            end if
18:        end if
19:    end for
20: end procedure
```

A new node (with an edge) is created at a level where an incoming packet does not match any existing node at that level in the graph (this needs to be checked for every level). Note that each node at level $n+1$ has two children nodes at level n because of the binary nature of subnetting operations. I, therefore, choose to initialize T_{n+1}[3], the retention period of nodes at level $n+1$, to the sum of retention periods of its children ($T_{n+1} = 2T_n$, where $T_0 = R$ a constant value set by an administrator). Upon creation of a node (with an edge), a default retention duration (*i.e.,* $T_n = 2^{n-1}R$) of its corresponding level n is assigned to the new node and edge. One may consider lager factors for the initial retention value (*e.g.,* $3^{n-1}R$) set for levels of the hierarchy. I note that longer retention period can result in improved visibility, but at much higher computing costs since a larger number of states need to be maintained. If the incoming packet matches an existing node and edge, then the corresponding retention duration is re-initialized.

Upon detection of an attacker node (at level n), the retention period of the two child nodes (*i.e.,* at level $n-1$) gets updated to the same value of the level n. I double the retention period of children for longer monitoring.

Detecting External Scanners & Flooders

I now show how my proposed model can detect scanners, especially those with low rate probing activity. Let's assume a simple threshold N is employed to detect scanners. A scanner node with retention duration T_n can be detected if it probes α internal hosts per epoch time and the condition (5.1) below is satisfied.

$$\alpha T_n \geq N \tag{5.1}$$

Detecting a slow scanner: A slow scanner can go undetected since its node/edge is removed from the graph every period of retention before hitting the threshold N. Instead, a higher level node with a larger (*i.e.,* power of two) retention duration T_{n+1} will be flagged as a scanner, and thereby the retention duration T_n of the child nodes gets updated. The time needed to detect the scanner child node is given by:

[3]A typical unit for duration value (such as T) is second.

$$2\frac{N}{\alpha} - T_n \tag{5.2}$$

Proof of equation 5.2. It takes $\frac{N}{\alpha}$ for the retention policy of a scanner node to get updated to a higher value of its parents (*i.e.,* time taken to detect a parent/root node as attacker). Since the scanner node has already had αT_n edges with internal hosts, it needs to accumulate $N - \alpha T_n$ more edges before being detected. This takes $\frac{N}{\alpha} - T_n$ with the probing rate α. Thus, in total, it take $2\frac{N}{\alpha} - T_n$ for a successful detection at the level of scanner node.

This process will be sequentially passed to nodes at lower levels until a successful detection at the lowest level ($n = 1$) is achieved. Hence, given the simple detection criteria (*i.e.,* condition (5.1) above), an external scanner can be detected within a guaranteed time t_{detect}:

$$t_{detect} = \begin{cases} \frac{N}{\alpha}, & \text{if } T_1 \geq \frac{N}{\alpha} \\ n\frac{N}{\alpha} - \sum_{i=1}^{n-1} T_i, & \text{if } T_{n-1} < \frac{N}{\alpha} \& T_n \geq \frac{N}{\alpha} \end{cases} \tag{5.3}$$

Proof of equation 5.3. If the probing rate is high enough for the host being detected within its initial retention period, then the detection time is $\frac{N}{\alpha}$. Otherwise, t_{detect} is derived by aggregating multiple processes defined in equation 5.2 till the lowest level.

Detecting distributed scanners: If a scan is performed by k scanners each having rate α that cannot be detected at host-level by default T_1 (*i.e.,* $\alpha T_1 < N$), it is possible (under certain conditions) to detect them at host-level earlier than $n\frac{N}{\alpha} - \sum_{i=1}^{n-1} T_i$, as computed in equation (5.3). In the best-case scenario, if all scanners are immediate neighbors of each other in Fig. 5.7 (*i.e.,* quickly converge to one node at higher level), and a root node (a parent covering all scanners) at higher levels has a sufficiently large retention period, it takes t_{detect} given by:

$$t_{detect} = \frac{N}{\alpha} + \sum_{i=1}^{n-1} (\frac{N}{2^i \alpha} - T_i) \tag{5.4}$$

Proof of equation 5.4. Detection time t_{detect} is derived in a similar way as in equation 5.3, except that the scanning rate for a node at level n is $2^{n-1}\alpha$ instead of α due to aggregation. Since the latter case is derived only by aggregating positive detection times from each level, it is always a positive value.

Otherwise, if the root node cannot be detected within its default retention period, then the detection process takes longer since the root node needs to be first detected as a scanner (by updating the retention period from upper layers). In this scenario, the detection time t_{detect} is given by:

$$t_{detect} = \frac{N}{\alpha} + \sum_{i=1}^{-1+\log_2 k} (\frac{N}{2^i \alpha} - T_i) + \sum_{j=\log_2 k}^{n-1} (\frac{N}{2^{-1+\log_2 k} \alpha} - T_j) \tag{5.5}$$

Proof of equation 5.5. Recalling that k is the number of scanners at the host-level, from level 2 to level $\log_2 k$, all children of a scanner parent are scanners, while from level $1 + \log_2 k$ to level n, each parent scanner has only one scanner child as explained in scenario for equation 5.3. Thus, the detection time is a combination of both processes.

To summarize, a scan can be detected at the host level within the duration t_{detect} given by equation 5.6, while the best-case is that all scanners are immediate neighbors with high scanning rate, and the worst-case is when scanners are located sparsely on the graph each with low scanning rate. In other words:

$$\frac{N}{\alpha} \leq t_{detect} \leq n\frac{N}{\alpha} - \sum_{i=1}^{n-1} Ti \tag{5.6}$$

where, n is the level at which the scan is first detected (*i.e.,* $\alpha T_n \geq N$ and $\alpha T_{n-1} < N$).

Proof of equation 5.6. The best-case scenario is when the scanning attack is detected (at the host level) within default retention period, while the worst-case is that all attackers can not be detected within default retention period and they are sparsely distributed.

Detecting distributed flooders at aggregation level: My proposed model can detect flooders at the highest aggregation level (*i.e.,* root node) when a group of flooders is involved. This enables us to effectively detect (and mitigate) distributed attacks.

I define a simple threshold N for detecting flooder hosts, similar to that in condition (5.1) but for the rate of incoming queries. If an external host queries an internal node with more than N packets per time epoch, it gets detected. Considering aggregation, a node at level n is detected as a flooder if its query rate α exceeds $N * 2^{n-1}$. Hence, given k immediate neighbor flooder hosts with attack rate α, my data structure is able to detect distributed attacks at the highest aggregation level n_{detect} given by:

$$n_{detect} = 1 + \log_2 k \tag{5.7}$$

Practical Design Choices

I have demonstrated the efficacy of my mathematical-based model to detect challenging (low-rate and/or distributed) scans and floods. To realize and further improve this model for detecting volumetric attacks in real networks at scale, I consider three key design choices as follows.

First, instead of aggregating external source entities sequentially by subset mask, I track their activities in my model using a **three-level hierarchy** including AS level, subnet level (*i.e.,* registered subnets under the administration of each AS), and host level (*i.e.,* IP address of individual external source hosts). Second, since applying thresholds only on the traffic rate of an external host cannot isolate low-rate attackers from normal users, I use a **collection of attributes** to accurately model the behavior of external hosts individually as well as at aggregated levels (*i.e.,* subnets and autonomous systems), as explained in §5.4.2. I replace the thresholding-based detection function with anomaly detection techniques. To enhance resilience against morphed attacks that deviate from known signatures [13], I train my models **only** with the behavior of benign external entities (*i.e.,* hosts, subnets, and ASes) and hence detect anomalies as described in §5.4.3. Third, to scale my solution it is important to react quickly, and manage costs of memory access efficiently since a large amount of data needs to be processed in real-time. I, therefore, use online algorithms [270] to receive one data point at a time and use it to update a set of attributes – the required statistics (variance and average) are computed in a single pass when costs of memory access dominate those of computation. I compute the variance attribute using Welford's method [271] and the average attributes are computed by exponential averaging. After the update, the data point is discarded and only the updated attributes are kept in memory.

5.4.2 Identifying and Computing Attributes of DNS Traffic for External Entities

I analyzed the DNS query behavior of external hosts (with their subnets and autonomous systems) in §5.3.2. Given the insights obtained from real attackers, I now identify and compute important attributes (for each external entity) needed for my detection models to distinguish normal external entities from anomalous ones.

Attributes

I showed in §5.3.2, DNS-based attackers tend to craft query packets using a set of predefined domain names. I define my first attribute as ***varPktSize*** (*i.e.,* variance of packet size sent by the external entity), since the size of query packets sent by scanners and/or flooders are less variant compared to queries from normal (legitimate) hosts.

Normal external users (including individual clients and recursive resolvers) may only query a limited number of internal hosts (*i.e.,* DNS servers of the organization). Scanners, on the other hand, contact a larger number of internal hosts (especially at relatively large time-scales). I therefore choose my second attribute as ***numHostQry*** (*i.e.,* number internal hosts queried by each external entity).

Both external flooders and heavy scanners send a much larger number of query packets (in total) to the enterprise network compared to normal external users. Flooders focus on one or a set of internal hosts, whereas scanners sweep over a wider range of internal hosts. So, I choose my third attribute as ***avgPktCountHost*** which is the average number of query packets sent to each internal host contacted.

For my last attribute (*i.e.,* ***varPktCountHost***), I compute the variation of query packet count sent to each internal host by an external entity. Note that the value of this attribute is smaller for scanners compared to flooders and legitimate users, since scanners tend to send an identical number of query packets (*e.g.,* one or two) to internal hosts of interest.

High-profile DNS query-based attacks can be quickly detected by either *numHostQry* (for heavy scanners) or *avgPktCountHost* (for heavy flooders). For relatively low-profile attacks (*i.e.,* distributed floods and/or slow scans), I need an enhanced visibility of the behavior of external entities using the collection of four attributes, aggregated-level models with dynamic retention policies. Next, I discuss how these attributes for each external entity (*i.e.,* node in the graph model) are computed in real-time.

Managing States for Edges

As explained in §5.4.1, an external entity is related to an internal host using an edge in my graph model. I dynamically update the four attributes of each external entity every epoch time (*e.g.,* one minute) using a number of states maintained for the graph edges.

For an edge, I track three main states including total number of packets (denoted by N_p), total volume of packets (denoted by V_p), and variance of packet size (denoted by σ_p) during each epoch – arrival of a DNS query updates these three states for the corresponding edge. Edge states are exponentially averaged (with weighting factor of 0.9) at the end of each epoch – I employ the Welford's method [271] to compute σ_p in a single pass (*i.e.,* online algorithm).

Computing attributes: Given edge states, the four attributes of an external node are computed as follows. *numHostQry* equals the number of edges associated with the node; *varPktSize* equals the weighted average of σ_p across all edges (*i.e.,* $\Sigma\frac{\sigma_p.N_p}{\Sigma N_p}$); *avgPktCountHost* equals the average N_p across all edges (*i.e.,* $\frac{\Sigma N_p}{numHostQry}$); and, lastly *varPktCountHost* can be derived by computing the standard deviation of N_p on all associated edges.

5.4.3 Anomaly Detection Model

I now train, tune and validate the accuracy of three anomaly detection models, namely host-level, subnet-level and AS-level for external source entities. I note that the intended pattern of inbound DNS traffic can vary across different enterprises, depending on the richness and size of their infrastructure, and their services offer-

Table 5.2: Summary of data cleansing for the university network (1 May 2018).

Reason of Removal	Number of Packets
Unanswered	9,970,082
NameError	689,422
ServerFailure	40,247
Refused	29,155
NotImplemented	21
FormatError	18

ings. Therefore, each network would have its own set of models trained by their own records of DNS traffic activity to achieve the best detection performance. To make the training process portable across enterprises, my engines for data cleansing, producing training set, and generating models are fully automated and consume DNS logs (*i.e.,* PCAP files) as inputs.

Dataset Preparation

Benign instances obtained from real DNS traffic of each enterprise are used for training anomaly detection models of the corresponding network. Additionally, I generate and collect data of DNS attacks including scans and floods of varying rates in my lab testbed. This attack dataset (together with the benign dataset) is used to evaluate the accuracy of my host-level anomaly detection.

Benign dataset: I clean my raw dataset (of 1st May) by removing unanswered and invalid queries, and use it for generating benign instances. I acknowledge that my cleaned dataset can still contain "not purely benign" instances, thus I tune a hyper-parameter called "contamination level" during model training to reduce the effect of outliers in the dataset. For example, I extracted 3.6M DNS queries (as benign) after removing 10.7M queries from the university dataset[4] on 1 May – details of removed queries are shown in Table 5.2. Using cleaned data, I generated 32.4M,

[4]I omitted results of data cleansing and model training for the research institute. Fairly similar observations were made in both organizations.

24.5M, 11.1M benign instances (of 1-minute granularity) for the host-, subnet-, and AS-level models respectively.

Attack dataset: I set up an isolated testbed in my lab to emulate an enterprise network communicating with external hosts via a border router. The internal and external networks were configured with a subnet from a /16 IPv4 address prefix. I configured a DNS server (running BIND 9) and one regular host inside the enterprise zone, and 3 DNS servers and 2 attacker machines (running a customized script using the Python Scapy library) on the external zone – each machine running Ubuntu 16.04.4 and equipped with a 2.1GHz CPU and 8GB RAM.

My attack script running on the two machines (*i.e.,* M1 and M2) generated query scans (from M1) to the entire IP range and query floods (from M2) on the DNS server of the internal zone (enterprise network). I generated a diversity of attack patterns by varying three parameters for each attack type. For query scans, I varied parameters as follows: the query rate from 1 to 72K (in 12 steps) packets-per-hour; query names selected randomly from a varying size (1 to 10) of a pre-populated list of the university sub-domains; and, the number of queries per internal host from 1 to 4. For query floods, the rate varied from 1 to 300 (in 12 steps) packets-per-second; query names were selected similar to query scan attacks; and, the query types were chosen from four common types (*i.e.,* `A`, `AAAA`, `ANY`, `PTR`). In total, I generated 480 scans and 480 floods with each attack lasting for 3 hours. Note that multiple attacks, each sourced from a unique crafted IP address, were scheduled in parallel. I collected data of the attacks on the testbed, computed attributes of external attackers during their activity, and generated 556,258 instances at host-level and 13,904 instances for both subnet- and AS-level – all instances associated with one subnet and one AS.

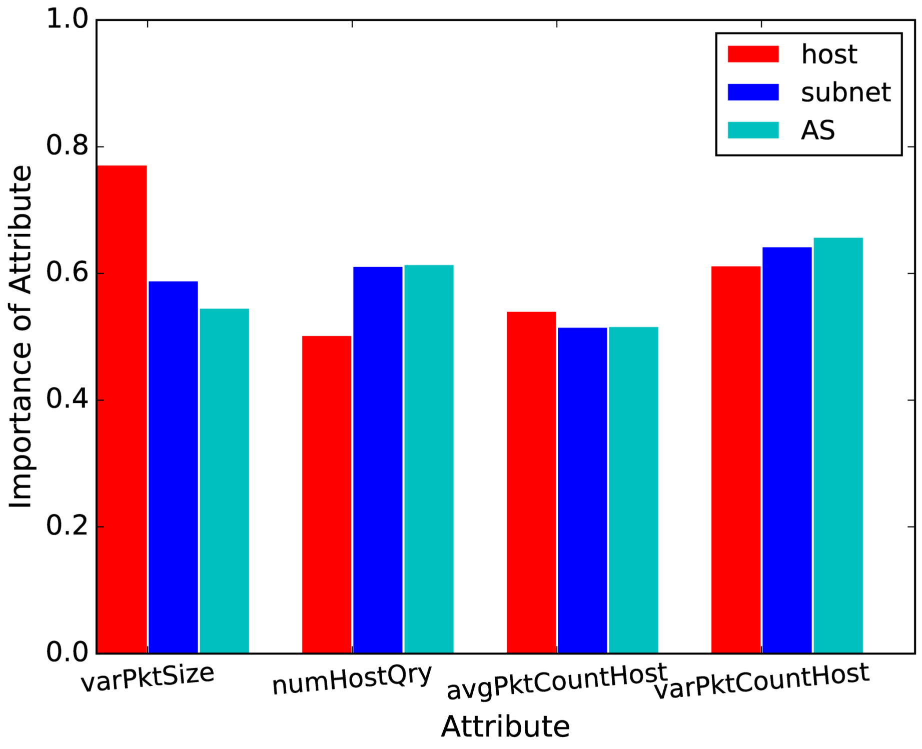


Figure 5.8: Importance of attributes across the three models.

Attribute Analysis

I now employ "information gain" (IG) metric to quantify the impact of my attributes in distinguishing benign and anomalous instances. This metric measures the correlation between each of the attributes and the predicted output of decision trees (valued between 0 and 1, where 0 indicates an irrelevant attribute and 1 highlights an important attribute). This method calculates the reduction of entropy values by excluding a certain attribute from prediction. I note that training a model with attributes of low information-gain can lead to issues like overfitting, since the classifier gets trained by noise or less-relevant information.

I show in Fig. 5.8 the importance of my four attributes for host-level, subnet-level, and AS-level models. It can be seen that the importance all attributes (across the three models of my hierarchy) is larger than 0.5, and hence carrying a significant amount of information in differentiating benign and malicious entities. In addition, I observe that ***varPktSize*** (with importance value equal to 0.772) is a fairly important attribute for the host-level model – this is because an anomalous external host will likely craft DNS packets of the same size, and hence resulting in a low (close to zero) variance for their packet size. Also, it is seen that the importance of ***numHostQry*** and ***varPktCountHost*** slightly increases from the host-level model to subnet-level and AS-level models, highlighting that these attributes become more influential at aggregate levels.

Model Tuning and Evaluation

I considered two widely used anomaly detection algorithms, namely *isolation Forest* [272] (a decision-tree based algorithm) and *one-class SVM* [273] (a high dimensional distribution-based algorithm). I evaluated the accuracy of these algorithms using 10-fold cross validation on the benign dataset (obtaining the True Positive rate) and testing on the attack dataset (obtaining the True Negative rate). Note that True Positives indicate benign instances that are correctly classified as benign, and True Negatives are attack instances that are correctly labeled as attack.

To be more specific, I trained and evaluated my model for each algorithm by varying the contamination-level parameter[5] for the isolation Forest, and kernel functions (*i.e.,* linear, Gaussian, polynomial, sigmod, and RBF) for the one-class SVM. For each set of tuning parameters or functions, the training dataset (benign only) is randomly partitioned into ten equal size subsets. Of the ten subsets, nine are used as training data, and the remaining subset is retained as the validation data for testing the model. During the testing phase, benign instances from the single subset of validation is used to compute the rate of True Positive (TP) while the entire attack dataset is used to compute the rate of True Negative (TN). I found that the isolation Forest model outperforms the one-class SVM model by both TP and TN rates. Given the multi-centric distribution of my benign dataset (instances

[5]A value between 0 and 1 that indicates the fraction of anomalies (*i.e.,* not benign instances) in the training dataset.

Table 5.3: Model tuning (host-level).

Cont. Level	**Accuracy**	**TN**	**TP**	**AUC**
0.0001	99.39%	99.98%	65.35%	76.94%
0.0002	99.42%	99.96%	68.41%	89.34%
0.0005	99.54%	99.89%	79.45%	97.05%
0.0010	**99.76%**	**99.81%**	**97.21%**	**99.96%**
0.0020	99.57%	99.61%	97.43%	99.91%
0.0050	99.03%	99.04%	98.524%	99.74%
0.0100	98.03%	98.03%	98.55%	98.86%
0.0200	96.09%	96.05%	98.74%	97.05%
0.0500	90.16%	90.01%	99.13%	61.34%

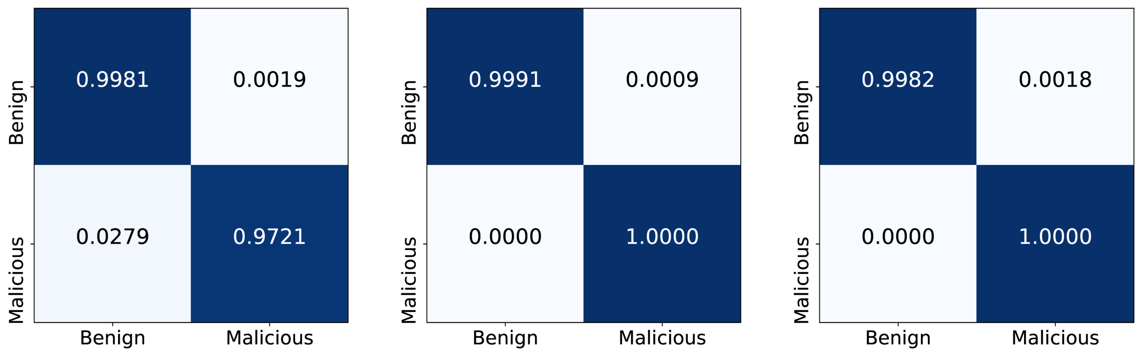
Table 5.4: Model tuning (subnet-level and AS-level).

Cont. Level	**TN of subnet model**	**TN of AS model**
0.0001	99.98%	99.98%
0.0002	99.97%	99.96%
0.0005	99.96%	99.90%
0.0010	**99.91%**	**99.82%**
0.0020	99.80%	99.63%
0.0050	99.02%	98.97%
0.0100	98.02%	97.94%
0.0200	96.08%	95.87%
0.0500	89.99%	89.91%

are geometrically located in several clusters [274]), the SVM model at best yields an overall accuracy of 63.6% (TN rate of 63.5% and TP rate of 74.6%).

I tuned and evaluated[6] the host-level model of isolation Forest considering both TP and TN rates, as shown in Table 5.3. I can see that increasing the contamination-level causes the TN rate to fall monotonically (from 99.98% to 90.01%), since the algorithm excludes more training instances at the boundary of benign clusters when the contamination-level gets larger. On the other hand, the TP rate is positively correlated with the contamination-level, as low-profile attackers become similar to benign hosts in their attributes. I, therefore, set the contamination-level to 0.001, resulting in the best overall accuracy among all models and a reasonably high rate

[6] In my evaluation, I used a mixed dataset (of labeled benign and attack instances) to compute overall accuracy, true negative rate and true positive rate.



(a) The host model. (b) The subnet model. (c) The AS model.

Figure 5.9: Confusion matrix of best-performing selected models at: (a) host, (b) subnet, and (c) AS, levels – columns correspond to true labels and rows correspond to predicted labels.

of both TN and TP.

By tuning and evaluating models of subnet-level and AS-level, I observe a similar impact of the contamination-level on TN rate (as shown in Table 5.4) – the TP rate would be consistently 100% (except for the contamination-level at 0.0001) since my attack instances at subnet-level and AS-level represent a network of high-profile attackers. Given the optimal value of the contamination-level (obtained from the host model), I achieve TN rates of 99.91% and 99.82%, and AUC values of 99.95% and 99.98% for the subnet-level and AS-level models, respectively. The three best-performing models trained with a contamination-level equal to 0.0001 are selected for my university dataset, and their performance is summarized by confusion matrices shown in Fig. 5.9.

By applying my best-performing models on the proposed graph data structure to the synthetic attack dataset, all external attackers are detected at the host level – 91.37% are detected immediately (within the first minute of their commencement); 3.45% (low-rate scans and floods) are detected in 2 minutes; and only 1.72% of very low-rate scans (with the rate of 1 packet per second) are detected after 8 minutes of their commencement.

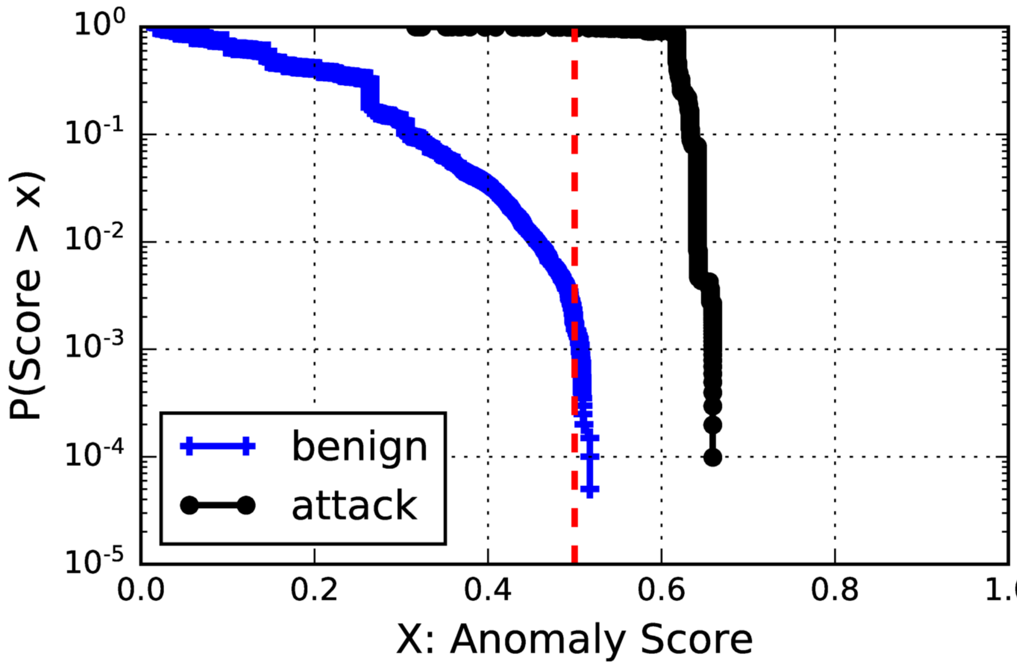


Figure 5.10: CCDF of anomaly score instances in evaluation dataset (host-level model).

Impact of training data composition on detection performance: I have so far generated an almost perfect isolation forest model (accuracy of 99.76%) by training it on the purified data (benign-only). It has been shown in [275] that fine-tuning the contamination-level (during training) may make no or little difference to the model performance in certain situations where the training data consists of well-distinguished malicious and benign instances (mix of "black and white" instances) with ground-truth labels. I, therefore, quantify the impact of impurified data (*i.e.*, raw DNS data, consisting of benign and malicious instances) on the performance of detection for the host-level model. I tune the contamination-level ranging from 0 to 0.2 is steps size of 0.005. I observe that the model gives its best overall accuracy of 93.33% (true positive 86.94%, and true negative 99.72%) at the contamination-level set to 0.085. This mis-detection of attack instances (low true positive) highlights the fact that inclusion of anomalies in the training data can be detrimental to the performance of isolation forest one-class classifier (the boundary of the model becomes loose), especially in absence of ground-truth labels for relatively uncertain benign and malicious instances ("gray" instances).

Understanding false alarms of my model: The isolation Forest algorithm outputs a score of anomaly (*i.e.,* a value between 0 and 1) for a given instance – where 0 score means purely normal and 1 indicates a definite anomaly. I use the anomaly score to quantify the severity of malicious behaviors for external entities – a high anomaly score indicates that the detected external entity strongly displays the behavior of an attacker (*e.g.,* high rates attack traffic or highly repetitive packet contents), whereas lower anomaly scores suggest that the anomalous external entity has slightly deviated from the expected normal behaviors (*e.g.,* a slow-rate scanner). I show in Fig. 5.10 the CCDF of anomaly score for my ground-truth evaluation instances (benign and attack). It can be seen that about 90% of benign instances (shown by blue lines and cross markers) have a low score of less than 0.3. I also observe that the anomaly score for only a small faction of benign instances (*i.e.,* 0.2%) exceeds the red boundary line (*i.e.,* 0.5) in Fig. 5.10, separating benign and malicious entities. I found that the reason for benign instances getting larger anomaly scores was their *avgPktCountHost* attribute which was slightly higher, compared to other benign instances. For example, an external host (in the benign dataset) had sent 9,492 query packets (all responded with `NoError` flag) to 12 internal hosts in an hour – I am not able to verify if this host (and it's behavior) was illegitimate or not.

Moving to the distribution of anomaly score for attackers (shown by black lines and circle markers in Fig. 5.10), about 85% of attack instances result a score of more than 0.6. I also see a tiny fraction (2.8%) of attacks (floods and scans) have a score of less than 0.5 generating False Positive alarms – these instances mainly correspond to (a) the beginning of low-rate scans depending on the traffic rate, and (b) low rate floods with a diversified set of query names (*e.g.,* 10 query names). I note that these low-profile attacks may not be detected by the host model in a short timescale, but they ultimately get detected by the aggregated-level models (as explained in §5.4.1).

Limitation of my evaluation: Note that my training dataset (benign) was obtained from a "real" production network while the ground-truth attack traffic traces were collected from a lab testbed ("synthetic"), and hence the overall evaluation mix may not necessarily represent realistic traffic instances. For future studies beyond this thesis, one may want to consider a more comprehensive attack dataset, covering a wider set of volumetric DNS attacks (direct and/or distributed) that are collected from real networks.

Testing the Dataset with a Commercial Firewall

I replayed my mixed dataset through a typical enterprise next-generation-firewall (Palo Alto Networks firewall appliance PA-3020 [276]) which was configured using the vendor's official user manual [202]. This firewall generates *Host Sweep* and *UDP Flood* alerts for DNS scans and floods, respectively. I found that the firewall missed 83% of scanners and 16% of flooders (*i.e.,* in total **51% true positives**) in my dataset, according to its threat logs – note that all missed attackers had a low rate (*i.e.,* below 20 hosts every 10 seconds for scanners and 100 packet-per-second for flooders) of DNS traffic. Also, the firewall generated no alert for benign instances (*i.e.,* **100% true negatives**) .

Comparing these results obtained from the firewall with those in Table 5.3, it can be seen that my anomaly detection scheme performs much better than the firewall by considering the TP metric (*i.e.,* 97.21% in Table 5.3 versus 51% highlighted in the above paragraph). In terms of the TN metric, the performance of my scheme is very close to that of the firewall (*i.e.,* 99.81% in Table 5.3 versus 100% stated in the above paragraph). Note that my scheme raises malicious alarms for a tiny fraction (*i.e.,* 0.18%) of benign instances due to impurities considered for the training dataset.

Testing the Dataset with Pure Thresholding of My Hierarchical Structure

In order to evaluate the efficacy of my hierarchical structure (with appropriate parameters R, α, N from §5.4.1), I applied thresholds on the packets rate in my dataset (mixed benign and malicious). My thresholds are chosen according to industry best practices [202] (*i.e.,* 120% of maximum normal value). For detecting flooders, I set host-level, subnet-level, and AS-level thresholds on ***avgPktCountHost*** equal to 51, 64, and 82, respectively. Also, for detecting scanners, thresholds on ***numHostQry*** are set to 8, 13, and 15 at respective levels of the hierarchy.

My evaluation results show that all of the attacks (100%) are detected by AS-level and subnet-level thresholding, while 89.42% of scans and 64.75% of floods are detected at the host-level, resulting in an overall true positive rate of 77.08%. Also, considering the true negative rates, I observe 98.24%, 98.57%, and 99.18% at AS-level, subnet-level and host-level, respectively.

In summary, applying the thresholding method of my hierarchical structure yields 99.18% TN and 77.08% TP at the host level, while my anomaly detection model gives 99.81% TN and 97.21% TP at the host level (Table 5.3), and the legacy thresholding employed by a commercial firewall results in 100% TN and 51% TP (§5.4.3). This shows that my hierarchical data structure significantly improves the performance of thresholding methods, but it is less performant when compared with my anomaly-based approach which models more comprehensive dynamic behavioral profiles.

5.5 Implementation and In-the-wild Detection

In this section, firstly, I demonstrate the efficacy of my detection method using a replay of DNS traffic collected from the two enterprises over a month. I then draw insights into the severity of attacks detected with a closer look at examples of low-profile scans and distributed floods. Next, I compare the output of my system using a public blocklist from Symantec and a commercial firewall from Palo Alto Networks. Finally, I quantify the performance of my system and show how I can detect DNS attacks in real-time with acceptable memory and CPU footprints for a one month period with real traffic of a large enterprise.

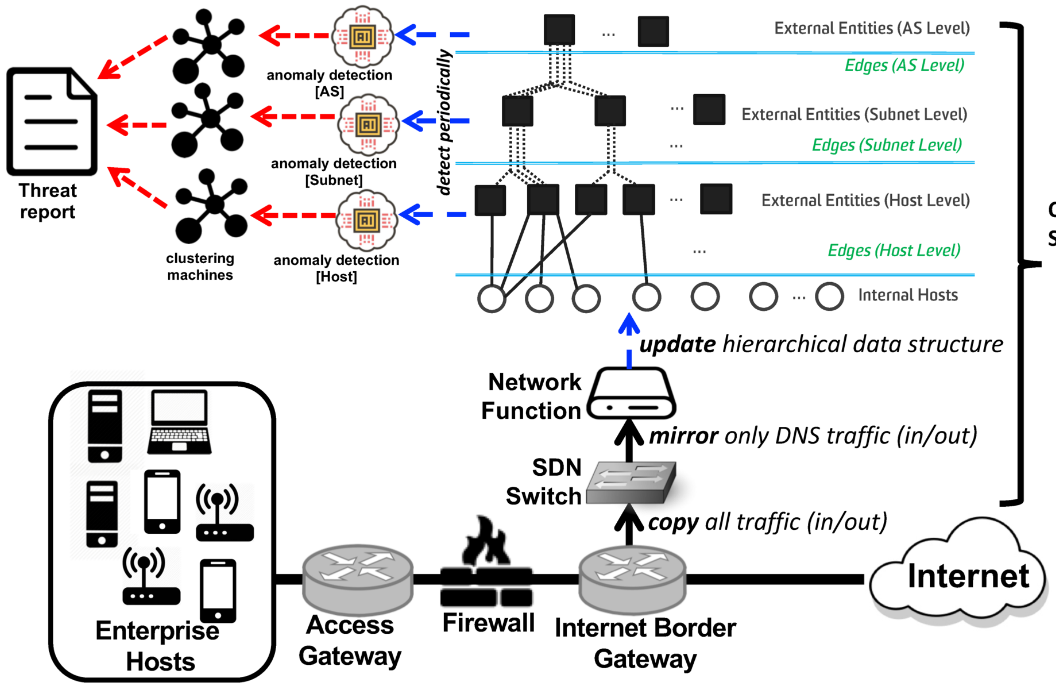


Figure 5.11: Prototype implementation.

5.5.1 Prototype Implementation

Fig. 5.11 depicts the prototype implementation of my real-time system that is deployed in my lab processing full DNS traffic copied from the border of both enterprise networks. I use an SDN switch (NoviFlow 2122 [240]) that takes a full copy of enterprise Internet traffic (both inbound and outbound) and selectively mirrors only DNS traffic (source or destination port number equal to 53) to a compute node (*i.e.*, network function). All software modules are implemented on a generic server (equipped with 16 2.10GHz CPUs and 64GB RAM) running Ubuntu 16.04.4. My packet processing module (*i.e.*, network function) written in the Golang language using the Data Plane Development Kit (DPDK) and the Intel NFF-Go library. It extracts necessary attributes (*i.e.*, source/destination IP/port and packet size) from incoming query packets, and passes them to update my hierarchical data structure also written in Golang. I use IPASN data files (*i.e.*, ".dat" files) to map external IPs

Table 5.5: Summary of in-the-wild dataset – May 2018.

	# Qry. in	# Resp. out	# Resp. in	# Qry. out	# ext. host	# ext. subnet	# ext. AS
University Campus	520,229,825	219,095,708	629,058,944	709,593,940	374,348	81,778	25,263
Research Institute	351,733,547	126,321,729	178,292,559	212,671,912	179,568	51,389	18,819

to their corresponding subnets and ASes, which are generated from MRT/RIB BGP archives [277]. My proposed detection modules are implemented using Python3 to be compatible with machine learning utilities. For decision-making functions, I use the best-performing anomaly detection models (from §5.4.3) trained with the contamination-level equal to 0.001 (at all three levels: host, subnet, and AS) – they perform reasonably well by both TP and TN metrics.

5.5.2 Summary of Dataset

A summary of my in-the-wild dataset collected over the month May 2018 is shown in Table. 5.5. For the university network, I have a total of 2.0B DNS packets including 520M incoming queries, 219M outgoing responses, 709M outgoing queries and 629M incoming responses. Focusing on external entities who sent DNS queries to the network, I see a total of 374K unique hosts associated with 81K subnets and 25K ASes. For the research institute, there were 867M DNS packets of which 351M packets were incoming queries sourced from 179M external hosts associated with 51K subnets and 18K ASes on the Internet.

I replayed the dataset of each organization on my system with the corresponding models. Instances were created at run-time for trained models to predict whether they are normal or anomalous. I made a log of all anomalous instances detected by my system for post-analysis and drawing further insights. After evaluating the university dataset, I found 14785 external hosts (*i.e.,* 3% of total hosts), 7403 subnets, and 2415 ASes flagged as anomalous. Also, for the research institute, 4332 external hosts, 2121 subnets, and 922 ASes were detected as anomalous entities.

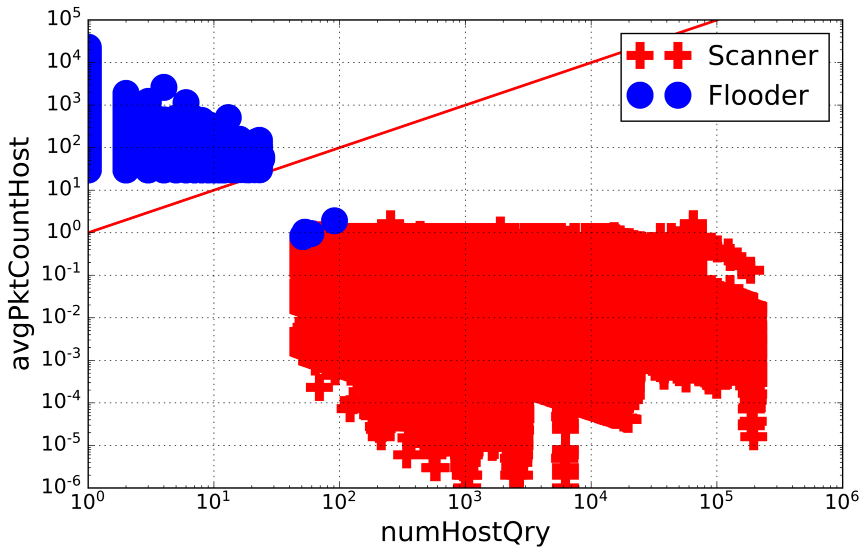


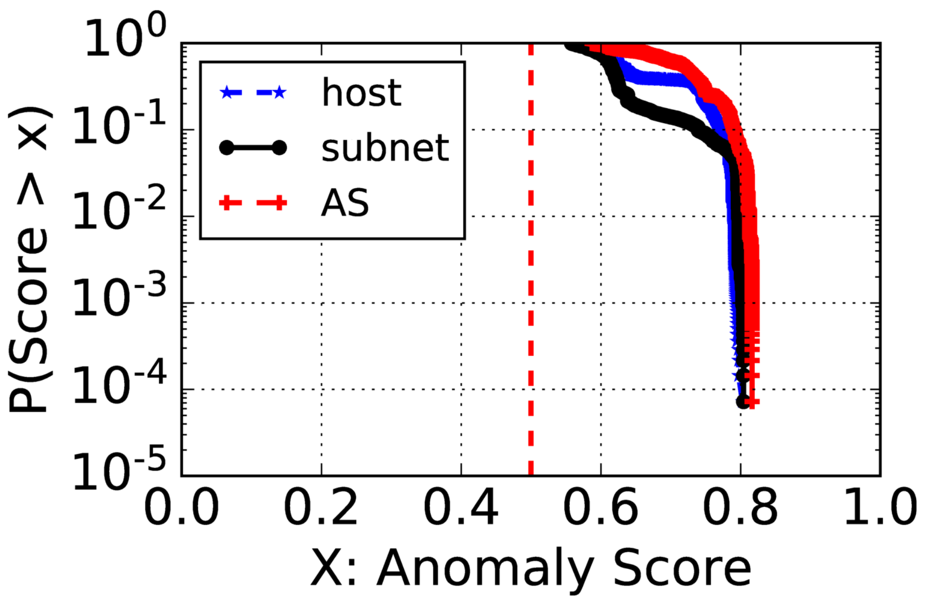
Figure 5.12: Clustering external anomalous hosts.

5.5.3 Clustering Anomalous Entities

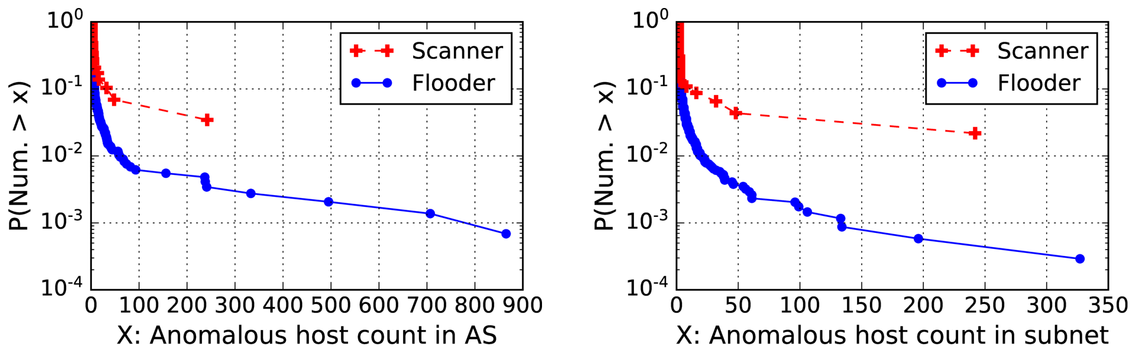
In order to distinguish scanners from flooders (at all three levels), I applied an unsupervised clustering algorithm, *i.e.,* expectation-maximization (EM), to those entities detected as anomalous. For the clustering model, I used two of my previously identified attributes (in §5.4.2) namely *avgPktCountHost* and *NumHostQry* as they collectively distinguish flooders from scanners.

As a result of clustering for the university network: 14171 flooders and 621 scanners were found at the host-level; 7493 flooders and 107 scanners were identified at the subnet-level; and, at the AS -level, 2430 and 47 flooders and scanners were found. For the research institute, 4332 hosts, 2122 subnets, and 921 ASes were identified as flooder entities, and also 490 hosts, 63 subnets, and 36 ASes were found as scanners.

Fig. 5.12 shows the scatter plot of two attributes, clearly separating flooders from scanners. As expected, flooders (shown by blue circle markers) are primarily located in the top left region of the plot while all scanners (shown by red cross markers) are grouped on the lower right region. Interestingly, I observe that several blue circles



(a) Anomaly score of anomalous instances.



(b) Anomalous host count in anomalous ASes. (c) Anomalous host count in anomalous subnets.

Figure 5.13: Severity of attacks in the university network: (a) anomaly score, and (b, c) distribution of clustered anomalous hosts in anomalous subnets and ASes.

(clustered as flooders) are located very close to scanners group – this is because their *avgPktCountHost* attribute value was higher than other scanners, and thus are classified as flooders.

5.5.4 Anomaly Scores

I now consider attack profiles by checking the anomaly score as well as the distribution of flooders and scanners at various levels of aggregation. Higher anomaly scores indicate a larger deviation from normal values of attributes (*e.g.,* large packet rates,

Table 5.6: Top scanner ASes.

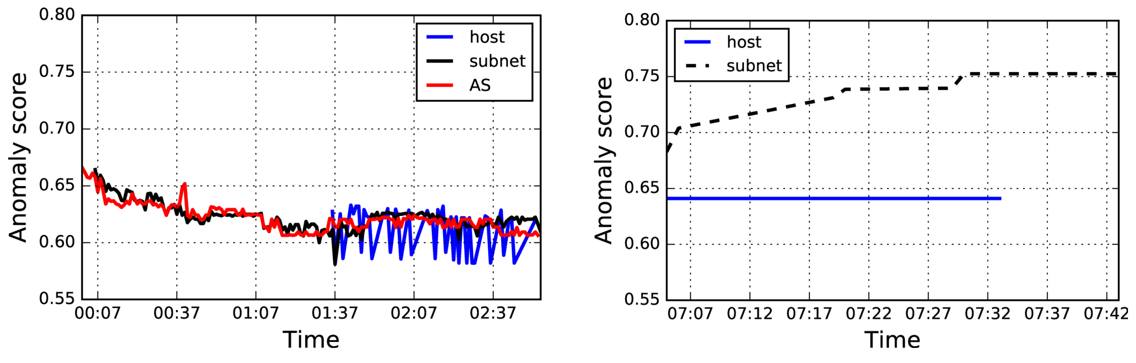
AS ID	**Loc.**	**Subnet**	**Host**
42570	CH	1	242
60781	NL	1	48
36375	US	1	32

Table 5.7: Top flooder ASes.

AS ID	**Loc.**	**Subnet**	**Host**
32934	US	11	865
16509	US	87	707
14618	US	59	495

highly repeated DNS query size, or numerous internal host contacted). Fig. 5.13(a) shows the score of anomalous entities detected by my isolation Forest models for hosts, subnets, and ASes. The first observation is that the anomaly score of detected attackers from real networks are relatively larger (*i.e.,* well above the border line 0.5) compared to attacks generated in my lab. I also see that at least 10% of instances (in all three models) have the score value greater than 0.7, highlighting the confidence of my models in detecting these anomalous entities in the wild.

I show in Figures 5.13(b) and 5.13(c) the distribution of clustered anomalous entities. Fig. 5.13(b) shows the CCDF plot of the number of anomalous hosts in anomalous ASes. The majority of ASes cover less than 100 anomalous hosts (both flooder and scanners). I observe that one AS has about 250 scanners but 7 ASes have a relatively large number of flooders (possibly distributed). The same observation is made for anomalous hosts of subnets, as shown in Fig. 5.13(c). I verified that the tail of curves in Figures 5.13(b) and 5.13(c) correspond to large anomaly scores (*i.e.,* between 0.75 and 0.80). The top distributed scanner hosts are located in one subnet, while the top distributed flooder hosts are spread across many subnets (under the administration of one AS). I list top scanner ASes and flooder ASes in Tables 5.6 and 5.7 respectively, based on their count of anomalous hosts. It can be seen that the top distributed scanner hosts are located in one subnet, while the top distributed flooder hosts are spread across many subnets (under the administration of one AS). Network administrators may use those alert lists to establish their local knowledge of external anomalous subnets/ASes and instrument security rules accordingly, *e.g.,* applying stricter access controls for external hosts from the suspected subnets/ASes.



(a) A low-rate scan. (b) A distributed flood.

Figure 5.14: Anomaly detection examples: (a) low-rate scan, and (b) distributed flood.

5.5.5 Two Representative Attacks

In order to demonstrate the efficacy of my scheme, I focus on two representative attacks (*i.e.,* a low rate scan and a distributed flood) that are typically missed by traditional solutions.

Fig. 5.14(a) shows the time-trace of anomaly score at all three levels, tracking the evolution of my detection. This attack is the low-rate scan which I manually identified in §5.3.3, shown in Fig. 5.3(c). As mentioned earlier, 37 hosts from a /16 subnet performed a scan of the university network simultaneously each with the rate of less than 1 packet-per-hour. I observe that this attack is first detected at the AS-level (shown by red lines in Fig. 5.14(a)). It takes 7 minutes for the subnet model to raise an anomaly alarm (shown by black lines in Fig. 5.14(a)), and the host model starts detecting attackers after more than an hour and a half (I plot the score for only one of 37 scanners as highlighted by blue lines in Fig. 5.14(a)). This detection was successfully achieved because of my hierarchical aggregation and dynamic retention policy (explained in §5.4.1) – detection of the attack at AS-level increased the retention duration of child subnets, leading to a detection at the subnet-level (with some delay) which in turn elongated the retention policy of child hosts, enabling the host model to detect the scanner hosts.

For my second example of attack, I show in Fig. 5.14(b) the time-trace of anomaly score for one /16 subnet consisting of 7 flooders which participated in a widely distributed flood that I described in §5.3.4, shown in Fig. 5.4(a). I note that this subnet is the only subnet under its AS that has anomalous behavior – the AS looks normal otherwise. I can see in Fig. 5.14(b) that right from the beginning of this flood (*i.e.,* around 7:07AM on 2nd May), the subnet was detected as a heavily anomalous entity, while its AS was classified as normal. I observe that the flooder host (one of 7) is consistently flagged as anomalous with the score 0.64 due to its repeated flooding pattern, while the anomaly score of the subnet rises in time as more external hosts join this distributed attack.

5.5.6 Comparison with Blocklist and Commercial Firewall

I selected 200 hosts[7] , those that are flagged during the entire month May 2018 – the top 100 hosts with the highest anomaly score and the bottom 100 with the lowest anomaly score (above the border line 0.5). I checked these hosts against an IP reputation repository (*i.e.,* blocklist) maintained by Symantec [278]. This web-based tool takes an IP address as input and reports if it was involved in malicious activities such as sending spam or spreading viruses. I found that the majority (*i.e.,* 63%) of hosts in my top 100 list are flagged as malicious IPs in the Symantec blocklist – 6 of them are scanners and 57 of them are flooders. Also, 58 hosts in my bottom 100 list are seen in the blocklist – all of these hosts are flooders.

Finally, to compare my system with a commercial firewall, I replayed my in-the-wild dataset through the Palo Alto Networks firewall appliance PA-3020 [276]. I extracted and analyzed the syslog file produced by the firewall during my traffic replay. The firewall detected 70 scanner IP addresses for the university dataset and 107 scanners for the research institute – this is a subset (11.3% for the university

[7]Automatic lookup of all flagged hosts in the blocklist is prevented by anti-robot image test.

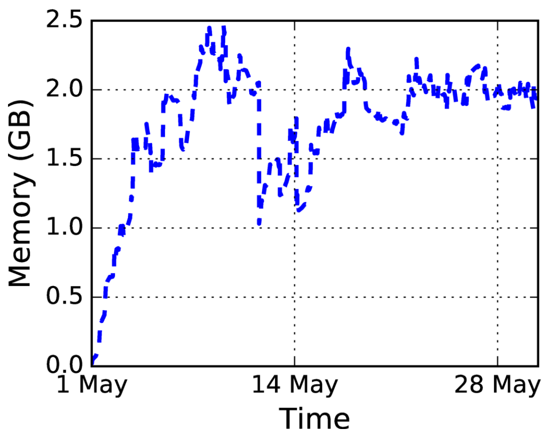
dataset and 21.8% for the research institute) of my detection results. Unsurprisingly, all of the scanners detected by the firewall had a high rate of probing, while none of the low-rate scanners were flagged.

For the query floods, the firewall captured 5 distributed attacks in the university network and two attacks in the research institute, as those victim internal hosts received an excessive number of UDP packets within a short time interval. Although the firewall logged all external IPs that sent packets when the alarm was triggered, it was not possible to **precisely identify and locate attackers**. My system, instead, not only detected all those distributed attacks flagged by the firewall, but also precisely captured the source (*i.e.,* external anomalous hosts, subnets or ASes). Besides, it is important to note that my system detected nine flooding attacks (sourced from the bottom 100 list) for the university network, but none of them were alerted by the firewall as the attack rate was relatively low. I manually checked these low-rate floods and found that they all sent repeated queries (with identical query name) to six non-DNS servers, two authoritative name servers, and three internal recursive resolvers.

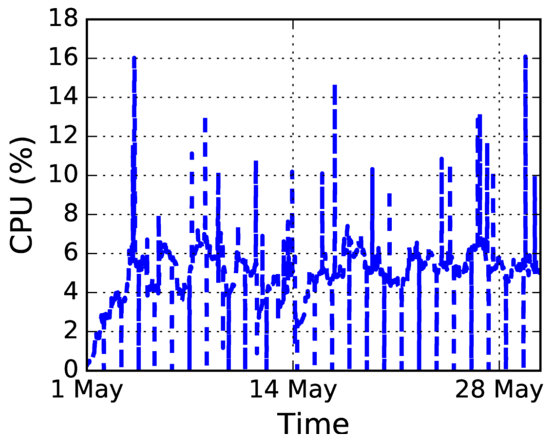
5.5.7 Real-Time Performance of My Detection System

To demonstrate the practicality of my proposed scheme I quantify the performance of my system (explained in §5.5.1) with one month's worth of real traffic from the university network[8]. Fig. 5.15 shows the real-time utilization of memory and CPU in my prototype. The memory consumption is recorded every minute, as shown in Fig. 5.15(a). Also, I measure the CPU utilization for two separate processes, namely *Updating* the graph structure upon arrival of DNS packets (Fig. 5.15(b)), and *Extracting* attributes from the graph structure and *calling* models at three levels of hierarchy (Fig. 5.15(c)) – each process utilizes one CPU. Additionally I

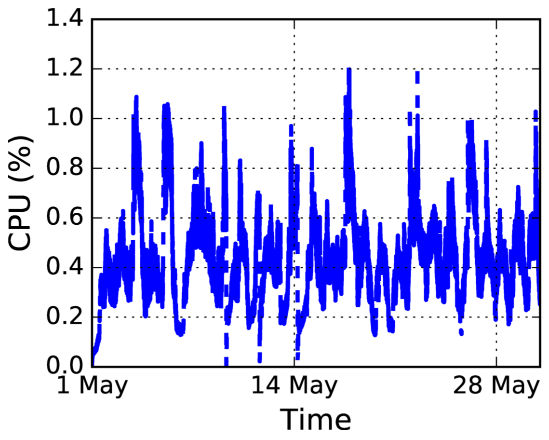
[8]I omit results of the research institute since its load was lower than the university network.



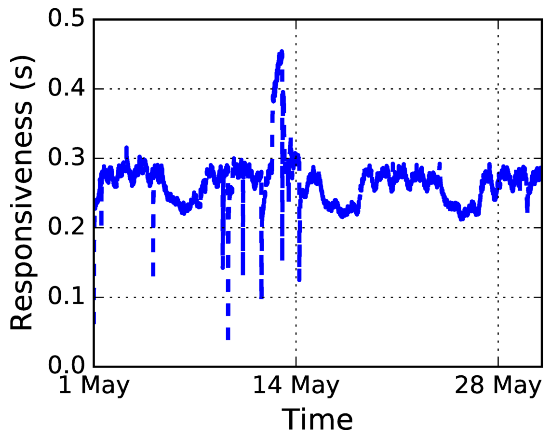
(a) Total memory usage.



(b) Updating the graph structure.



(c) Extracting attributes & calling models.



(d) Prediction responsiveness.

Figure 5.15: Real-time performance of my detection system under full load of the university campus network.

quantify the inference responsiveness (*i.e.*, time taken for computing attributes and obtaining results from models at the end of each epoch) as shown in Fig. 5.15(d). It is seen that all four metrics are bounded by reasonable values: the memory consumption is capped at 2.5GB, Updating and Processing respectively use up to 16% and 1.3% of CPU, and the system responds in less than 0.5s. Note that my system performance fluctuates (within a bounded region) due to the variation in the traffic. This demonstrates that my system can meet reasonable performance criteria typically required by enterprise network operators.

5.6 Conclusion

Enterprise networks are the target of sophisticated DNS attacks in the form of query floods, reflection and amplification attacks, and scans. Existing security appliances are not well-equipped to protect network assets from dynamic attacks sourced from distributed and automated external hosts on the Internet. I have developed a hierarchical method using anomaly-based detection models to automatically detect DNS floods and scans of varying rates sourced from one or a distributed set of external hosts. I highlighted the characteristics of malicious entities sending query-based attacks, developed a hierarchical and dynamic graph data structure for scalable monitoring and detection of scans and volumteric attacks, identified key attributes to effectively differentiate attacker entities versus normal external users, and employed anomaly detection models (trained/tuned using benign and attack traffic) in my dynamic data structure. Lastly, I demonstrated the efficacy of my scheme and compared my system with a public blocklist and a commercial firewall.

Chapter 6

PEDDA: Practical and Effective Detection of Distributed Attacks on Enterprise Networks via Multi-stage Progressive Inference

Contents

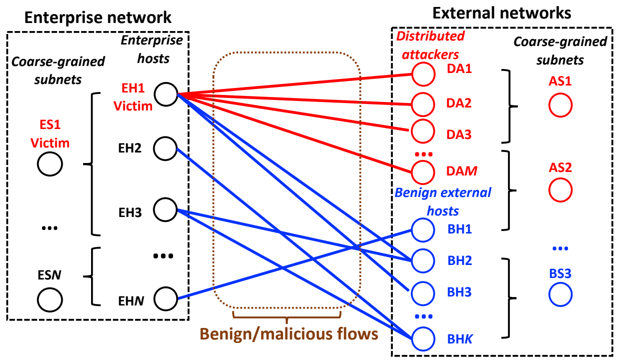
Network attacks on enterprises are becoming distributed in sources and versatile in patterns. However, practical solutions such as commercial firewalls are *ineffective* in detecting distributed sources and malicious flows. They often focus on enterprise assets (potential victims), given their limited resources for coarse-grained monitoring. In contrast, fine-grained flow-level detection algorithms proposed in academic research are *impractical* to handle high traffic rates with millions of concurrent flows in a large enterprise at scale.

To address the effectiveness and practicality of attack detection, I present PEDDA, a multi-stage progressive inference method that optimally selects detection stages of different visibility and orchestrates their granularity in accordance with the evolution of distributed attacks and available computing resources. **First**, I formally model the time complexity of traffic processing in legacy static solutions to highlight their performance bottlenecks. **Second**, I develop a progressive method to detect distributed attacks at multiple inference stages each with an orchestratable granularity, whereby packet streams are reactively partitioned and processed by different stages depending on the evolution of attacks. The granularity of each stage is dynamically orchestrated through optimization constrained by available computing resources. **Third**, I build a proof-of-concept prototype using three inference stages that monitor active enterprise hosts, isolate specific victims under attacks, and differentiate distributed sources and flows from benign instances. Evaluation results using real enterprise traffic and DDoS attacks show that PEDDA outperforms its counterparts in detecting distributed attacks at all stages with the finest granularity while practical for real-time deployment.

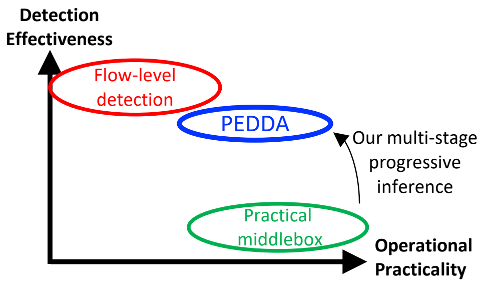
6.1 Introduction

Distributed network attacks targeting hosts in an enterprise such as DDoS [24, 78, 279–281] have reached an alarming rate with high frequency in occurrence, diversity in attack patterns and agility in bypassing countermeasures by security appliances [282–285]. A distributed attack usually exploits different protocols [24, 286] (*e.g.,* DNS, HTTPS) and uses large-scale botnet devices each starts with its own attacking strategy [44, 287, 288]. During attacks, malicious flows generated by distributed attackers are mixed with legitimate traffic from benign external hosts – a visual example is shown in Fig. 6.1(a). Therefore, effective detection of a network attack requires identifying victims from enterprise hosts, distributed sources from a large set of external hosts, and malicious flows from a high volume of concurrent connections.

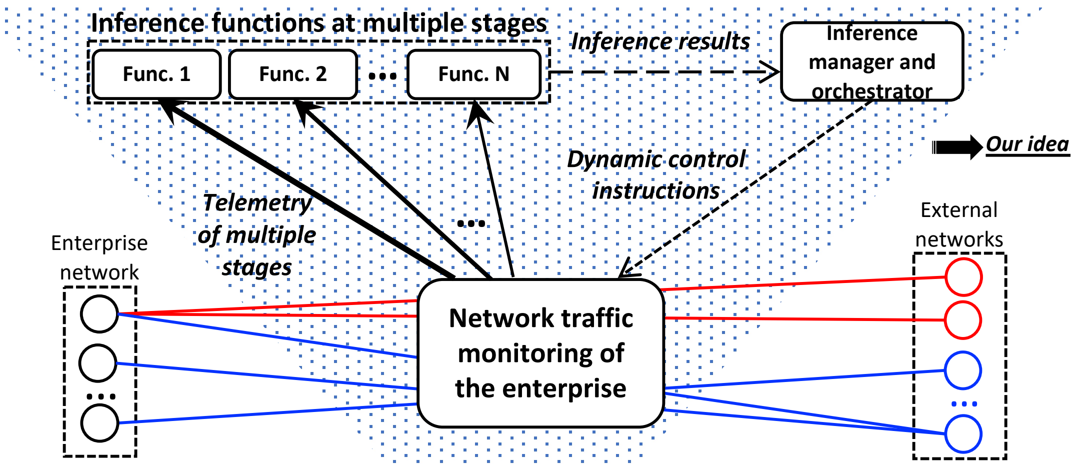
As the de facto solutions, firewall appliances and network intrusion detection systems (IDS) use static security signatures on pre-defined IP lists [202–204, 253]. Although some manufacturers such as Fortinet and Palo Alto starts to implement machine learning detection in their DDoS detection products, they still require configurations from IT operators based on their expertise to monitor certain hosts or coarse-grained subnets in their networks. Such approaches are scalable and can detect typical network attacks towards critical assets (as potential victims) specified in the policies. However, they are not able to differentiate external attackers and malicious flows versus benign instances since network telemetry of distributed external hosts and flows are rarely tracked for scalability concerns. Thus, the consequent attack mitigation often introduces collateral damage on normal communications. There are also effective attack detection systems using flow-level telemetry developed by research communities [64, 66, 75, 115], which can precisely isolate not only victims but also distributed sources and malicious flows. Such methods maintain flow-level telemetry between hosts, which provide fine-grained statistics for effective detection but incur high computational costs. Therefore, they are not applicable to large enterprise networks with high throughput of millions of concurrent flows.



(a) Anatomy of distributed attacks.



(b) Trade-off by legacy solutions.


Inference functions at multiple stages
Func. 1
Func. 2
Func. N
Inference results
Inference manager and orchestrator
Our idea
Dynamic control instructions
Telemetry of multiple stages
Enterprise network
External networks
Network traffic monitoring of the enterprise

(c) My idea of multi-stage progressive inference.

Figure 6.1: Introductory – (a) anatomy of distributed attacks, (b) trade-off between detection effectiveness and practicality, and (c) my idea of multi-stage progressive inference via dynamic control of programmable networks.

As visually summarized in Fig. 6.1(b), proprietary security middleboxes (the red ovals) are practical but not effective in identifying distributed attack sources and flows from benign instances, whereas detection methods on flow-level telemetry (the green ovals) are effective but not practical for high-throughput enterprise environments. Instead of accepting the trade-off, I see opportunities by programmable networks (*i.e.,* SDN and NFV). As an emerging paradigm, it has shown huge potential in making network defense functions and packet routing elastic and flexible at run-time. For example, *Bohatei* [11] proposes ISP-level run-time orchestrations of proprietary security middleboxes at fixed locations, *PSI* [232] instruments enterprise network traffic to its desired security appliances on-demand, and *Poseidon* [10] achieves flexible reconfiguration and updates of DDoS defense policies. Existing solutions show their promise in various aspects of flexible distributed attack detection and mitigation, which inspire my idea (visually shown in Fig. 6.1(c)) to address the dilemma between detection effectiveness and operational practicality for enterprise networks through a multi-stage progressive inference method.

In this chapter, I present PEDDA, a practical and effective method that detects distributed attacks through multiple inference stages (*e.g.,* victims, sources, and flows) progressively. To achieve detection effectiveness while holding practicality, PEDDA employs multiple inference stages with telemetry of different computational costs and orchestratable granularity (*i.e.,* fine-grained IP level or coarse-grained subnet levels) that are instructed by the reactive control of programmable networks. During operations, the received traffic is proactively processed by low-cost inference stages, whereas a high-cost stage reactively performs fine-grained inspection only on the packet streams partitioned by its prior stage. In addition, the granularity of each stage is dynamically orchestrated through an optimization problem constrained by available computing resources and complexities of the traffic. Note that I do not develop novel detection algorithms and functions, but propose a method that manages them at a progressive manner to achieve practical and effective detection. My proof-of-concept prototype uses three inference stages to monitor active enterprise hosts, detect victims with abnormal volumetric profiles, and identify distributed sources with flows of attacks from their benign cohorts. I make three specific contributions.

First, to motivate my PEDDA design, I model the traffic processing of legacy systems that matches packet streams, extracts packet information, maintains network telemetry, and makes detection inference. I mathematically formulate its per-packet CPU time consumption to highlight their bottlenecks in achieving both operational practicality and detection effectiveness.

Second, to address the identified bottlenecks, I design a multi-stage progressive inference method as the PEDDA architecture. It detects distributed attacks through multiple inference stages (*e.g.,* victims, sources, and flows) progressively. Those stages are with increasing computational costs as they are maintaining telemetry of higher complexity. During operations, the majority of traffic are processed by low-cost inference stage proactively, driven by the results from prior stages, the minor fraction of packet streams is inspected by high-cost inference stages for attack detection at further progression (*e.g.,* distributed sources and flows) through the dynamic controls of programmable networks (*i.e.,* NFV and SDN),. To avoid PEDDA being overwhelmed by the complex traffic compositions especially for high-cost stages, an orchestrator is designed to adjust the granularity of each stage from the finest IP level to aggregated subnet levels by solving a run-time optimization problem constrained by available computing resources.

Third, I realize PEDDA as a proof-of-concept prototype using a commodity server and an OpenFlow SDN switch that is ready to be deployed in a large enterprise. Driven by the insights obtained by an empirical traffic analysis of a representative enterprise, it uses three practical inference stages that detect active enterprise hosts, victims, and distributed attackers with flows progressively and reactively. The prototype is evaluated on real enterprise traffic traces injected with a public-available DDoS attack dataset and compared with state-of-the-art systems to highlight its supremacy in detection effectiveness and operational practicality.

Roadmap: I discuss the background and related works in §6.2, model and highlight bottlenecks of legacy detection solutions in §6.3, propose my multi-stage progressive inference method as PEDDA architecture in §6.4, realize and evaluate a proof-of-concept prototype with three inference stages in §6.5, and conclude this chapter in §6.6.

6.2 Background and Related Work

In this section, I highlight key requirements of enterprise network attack detection as suggested by both industrial and research communities in §6.2.1, and discuss related works in §6.2.2.

6.2.1 Key Requirements

I now explain the key requirements of an ideal distributed attack detection system for enterprise networks as suggested by the community, including telemetry visibility and inference precision for effective detection of attacks; scalability, automaticity, and deployability for practical operation in an enterprise environment.

Effective Detection

Distributed network attacks such as service probing (*i.e.,* host/port scans [29, 30, 38]) and denial-of-service (*i.e.,* DDoS [282]) toward enterprise assets has been evolved from a single source and protocol to more complicated forms – one attack may come from distributed malicious hosts (*e.g.,* compromised PCs and IoT devices [43]) and use various protocols (*e.g.,* SYN flood via HTTPS protocol or UDP reflection via Memcached port). Thus, to capture the diversified patterns of potential attacks, an ideal security solution should have a good **visibility** into traffic characteristics of all involved hosts and their network flows [289]. Besides, malicious traffic generated by attackers are likely to be mixed with legitimate flows from benign external hosts, which is particular true especially near the victim-side [9]. To sufficiently eliminate malicious traffic while not interrupting benign activities, detection systems are expected to be **precise** in differentiating malicious external hosts and flows of an attack from benign ones.

Practical Operation

To be practically operated in an enterprise network, an attack detection system should be **scalable** for high-throughput networks with several tens of gigabits per second and millions of concurrent flows. Besides, as highlighted in [2, 47], operators of large enterprises (*e.g.,* universities and research institutes) may not be fully aware of hosts connected to their networks. Thus, configuring proper security policies on IP addresses is challenging or even unpractical. A detection system would be more operable if it could automatically discover the enterprise hosts that need to be protected – I refer to this feature as **automaticity**. Lastly, existing network topology is reluctant to changes since any update on existing devices may incur high operational costs (*e.g.,* offline time, error rate, and labor effort) [267]. Therefore, a practical detection system is expected to be easily deployed (*i.e.,* **deployability**) at a single link rather than many vantage points of the protected networks.

6.2.2 Current Detection Solutions

State-of-the-art solutions for distributed network attack detection can be categorized into three groups, namely practical security middleboxes, flow-level attack detection methods, and prototypes using programmable networks. In what follows I describe their merits and gaps regarding the key requirements articulated in §6.2.1.

Practical Security Middleboxes

They are mature products in the market including next-generation firewalls (NGFW) and intrusion detection systems (IDS). Such systems are packaged as either proprietary hardware appliances (*e.g.,* CISCO [204], Fortinet [203], and Palo Alto [202] firewalls) or software tools (*e.g.,* Bro [97] and Snort [212]). They apply security signatures [290] (manually configured by IT operators or automatically obtained via

ML algorithms) such as thresholds on packet arrival rate for a pre-defined asset list to identify victims of network attacks. Those practical middleboxes take scalability and deployability as their top priorities [291, 292]. Thus, they barely support flow-level telemetry in practice that is necessary in identifying distributed attackers and malicious flows mixed with benign traffic. As a consequence, collateral damage to legitimate traffic (*e.g.,* dropping of benign packets [293]) is unavoidable during attack mitigation processes [9, 232].

Statistical Methods on Flow-level Telemetry

To achieve effective detection and mitigation of distributed attacks, researchers develop statistical methods leveraging fine-grained flow-level telemetry. Graph structures that profile network flows between hosts are frequently used such as in *BLINC* [64], *SpotLight* [75] and *AGM* [66]. By having detailed visibility into every single flow between external and internal hosts of the monitored network, distributed attackers and malicious flows can be detected and mitigated precisely. However, due to the high computational overheads [16] of complex flow-level telemetry, such methods are often applied to low-throughput networks of less than a gigabits per second [64] or a limited range of traffic types such as DNS [5] and HTTP [288] protocols.

Prototypes using Programmable Networks

Programmable networks (*i.e.,* SDN and NFV) make flexible and elastic traffic management possible through reconfiguration of network functions at run-time. Researchers have developed prototypes using these techniques to achieve objectives including real-time execution of user-defined network telemetries [86], detecting malicious flows from their first few packets via reactive routing [140], orchestrating defense middleboxes at fixed locations to handle attacks with changing patterns [11, 232], flexible network intrusion detection functions [113], and reconfigurable defense

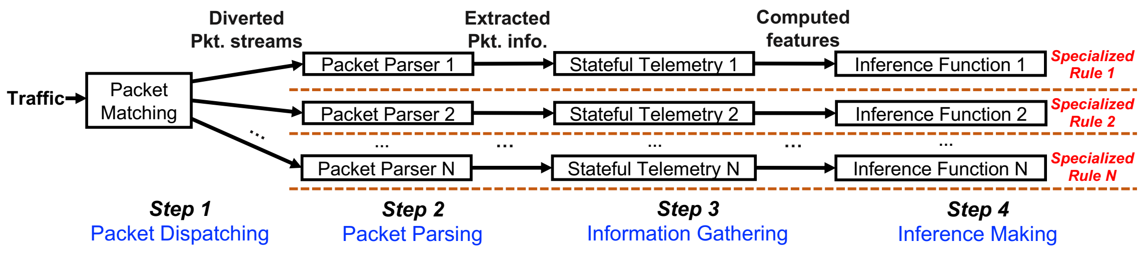


Figure 6.2: Traffic processing pipelines (that process received packet streams) of legacy solutions.

strategy for volumetric DDoS attacks [10]. Such works break the limitation of proprietary security systems that have fixed locations and detection capabilities through reactive control of programmable networks, which inspire my design of PEDDA to address the trade-off between effectiveness and practicality of distributed attack detection for an enterprise.

6.3 Analysis of Legacy Solutions

In this section, to motivate my design, I analyze legacy detection solutions to highlight the bottlenecks that hinder them from being both effective in attack detection and practical in operation. Specifically, I first model the traffic processing pipeline of legacy systems (in §6.3.1), and then mathematically formulate its per-packet CPU consumption (in §6.3.2) to identify the performance bottlenecks that could be further optimized.

6.3.1 Modeling Traffic Processing Pipeline of Legacy Solutions

Traffic processing of existing attack detection solutions can be modeled as a collection of independent pipelines as shown in Fig. 6.2. Each individual pipeline is responsible for one detection task (*e.g.*, a firewall policy configured by network operators) and consists of four steps to process packet streams, including packet dispatching, packet parsing, information gathering, and inference making.

Four Traffic Processing Steps

Packet dispatching is the first step that decide what further process(es) should be applied to each packet given its metadata. The second step (*i.e.,* packet parsing) extracts specific information from each packet (such as source and destination IP addresses, port number, and protocol) that is forwarded to the pipeline. In the third step (*i.e.,* information gathering), those extracted packet information will then be used to update stateful network telemetry maintained by each pipeline for its detection task. In the final step (*i.e.,* inference making), inference functions of each pipeline give detection results on attributes computed from its network telemetry.

A Demonstrative Example

Now I describe a walking example to help us better understand the detection pipelines. Assuming an enterprise network operator plans to protect its key website server on the IP address `a.b.c.d` against TCP-SYN based DDoS attacks.

A practical detection approach: The operator may configure a policy on its enterprise border firewall [203, 204, 293] to raise alarm if the monitored IP address `a.b.c.d` receives more than N TCP-SYN packets during a time interval T seconds [294]. After installation of this policy, all packets received by the firewall will be checked for their transport-layer headers for TCP-SYN flag, and destination IPs for `a.b.c.d` (*i.e.,* step 1). The matched packet streams are sent to the parser (*i.e.,* step 2) that extracts packet count as required by the following network telemetry. The telemetry (*i.e.,* step 3) maintains a total number of packets arrived on `a.b.c.d` during the current time interval . A corresponding inference function (*i.e.,* step 4) checks packet count from the telemetry every T seconds and raise alarms if the current value exceeds the threshold N.

An effective detection approach: If this task is performed by a flow-level detection method [64, 66, 75], the inference results can be more precise through a more complicated process. In the packet parsing step, instead of only reporting packet count to the stateful telemetry, all metadata that maps a packet to its associated flow are extracted, including transport-layer protocol, TCP flags, IP addresses and port numbers of source and destination. In step 3, the extracted information is used to update flow graphs (a complex telemetry) between the monitored IP `a.b.c.d` and external hosts. In step 4, an external host would be labeled as attacker if its flow profile appears malicious (*e.g.,* sending excessive number of flows and most of them are TCP-SYN packets). Although this method is able to identify sources and flows in distributed attack, as you will see in my following formulation, the per-packet time consumption of this effective method is too large to be operated at high-throughput enterprise networks.

6.3.2 Formulating Per-Packet CPU Consumption

I now mathematically formulate its per-packet CPU consumption[1] of traffic processing pipelines for legacy solutions (discussed in §6.3.1) to formally identify their performance bottlenecks. To be specific, I formulate the CPU time consumption for a packet arrived and processed by the four serial steps as the indicator of its computational complexity.

Let's use t to denote the CPU time consumption of a packet processed by the entire pipeline. Intuitively, t can be expressed as the summation of time consumption in each step as formulated by Eq. 6.1, where t_r, t_p, t_g, and t_m are the time consumption for packet dispatching, parsing, information gathering, and inference

[1]As reported by *NitroSketch* [230], CPU consumption is the most critical performance metric deciding the throughput of a networking system, thus, I formulate per-packet CPU consumption of pipelines in Fig. 6.2.

making, respectively.

$$t = t_r + t_p + t_g + t_m \tag{6.1}$$

I note that the inference making step for distributed attack detection is often executed periodically (*e.g.,* every 10 seconds in a typical firewall [293]) and not triggered for each packet. It is usually treated as a independent process and placed outside the packet processing pipeline. Therefore, I omit t_m in my following analysis. In what follows, I give my formulations for t_r, t_p, t_g, and t, respectively.

Packet Dispatching

I first formulate the time consumption t_r for each packet in the first step. A detection system has its network interfaces that receive packet streams. Each arrived packet is checked by a set of matching policies (or rules) to decide which parser(s) should process it. I assume that there are n_r packet matching policies and n_p parsers in the next step. The best case (*i.e.,* smallest time consumption) $t_{r,min}$ is achieved when a packet is matched by the first policy and only sent to one parser in the next step. While the worst case $t_{r,max}$ is met when the packet is checked for all n_r policies and sent to all n_p parsers. If I define the time consumption for a packet being checked for a policy is $k_{r,c}$ and being sent to one packet parser is $k_{r,p}$, the best and worst case can be derived as in Eq. 6.2.

$$t_{r,min} = k_{r,c} + k_{r,p}; t_{r,max} = k_{r,c}n_r + k_{r,p}n_p; \tag{6.2}$$

After denoting each policy and parser by their hit probabilities, the average time consumption $t_{r,avg}$ for each arrived packet in this step can be formulated as in Eq. 6.3, where $p_{r,i}$ is the hit probability for the ith matching policy, and $p_{p,i}$ is the probability for a packet being sent to the ith parser.

$$t_{r,avg} = k_{r,c}\sum_{i=1}^{n_r} p_{r,i} + k_{r,p}\sum_{i=1}^{n_p} p_{p,i}; \tag{6.3}$$

Packet Parsing

In this step, parsers extract the required packet information from each layer (*e.g.,* network layer, transport layer, and application layer) hierarchically [130]. A simple parser only processes the top layer while a costly one may extract data from all layers. Therefore, the CPU time consumption t_p for a packet in this step depends on the number of parsers involved (up to n_p) and number of packet layers to process by each parser. Assuming that the ith packet parser extracts data from a total of $n_{l,i}$ packet layers and the time consumption to process each layer is k_l, the total time consumption by the ith packets is expressed as $n_{l,i}k_l$. The best case $t_{p,min}$ is achieved when only the simplest parser is used while the worst case $t_{p,max}$ is reached when all parsers are utilized for a packet. The two cases can be formulated as in Eq. 6.4, respectively.

$$t_{p,min} = k_l \min_{i=1}^{n_p} n_{l,i}; t_{p,max} = k_l \sum_{i=1}^{n_p} n_{l,i}; \tag{6.4}$$

The average time consumption $t_{p,avg}$ for each packet is obtained by weighting each parser by its hit probabilities ($p_{p,i}$ for the ith parser), which can be mathematically shown in Eq. 6.5.

$$t_{p,avg} = k_l \sum_{i=1}^{n_p} p_{p,i} n_{l,i}; \tag{6.5}$$

Information Gathering

In this step, stateful network telemetry of each detection task may get updated by packet information from parsers. Telemetry are hold by data structures of each monitored entity (*e.g.,* hosts and flows). It can be a simple list with keys as the protected IP addresses and contents as their packet counts, or a complex attributed graph [66] tracking statistics of each flows [295]. Therefore, CPU time consumed for updating a telemetry depends on their structure types and current sizes. I use n_g and $N_{g,i}$ to represent the total number of telemetry in this step and current size of the i th telemetry, respectively. The CPU consumption of updating the i telemetry can be expressed as a fixed function $\theta_i(N_{g,i})$ of time complexity[2]. The best case $t_{g,min}$ is reached when only the simplest telemetry is updated by a processed packet, while the worst case $t_{g,max}$ is encountered when all n_g data structures are updated – I formulate them in Eq. 6.6.

$$t_{g,min} = \min_{i=1}^{n_g}(\theta_i(N_{g,i})); t_{g,max} = \sum_{i=1}^{n_g} \theta_i(N_{g,i}); \tag{6.6}$$

The average time consumption $t_{g,avg}$ by an arrived packet in this step is obtained by weighting all data structures by their probability as expressed by Eq. 6.7, where $p_{t,i}$ is the hit probability of the ith telemetry.

$$t_{g,avg} = \sum_{i=1}^{n_g} p_{t,i}\theta_i(N_{g,i}); \tag{6.7}$$

The Entire Pipeline

Now I summarize the best (t_{min} in Eq. 6.8), worst (t_{max} in Eq. 6.9) and average (t_{avg} in Eq. 6.10) per-packet CPU time consumption by aggregating the individual results of each step.

$$t_{min} = k_{r,c} + k_{r,p} + k_l \min_{i=1}^{n_p}(n_{l,i}) + \min_{i=1}^{n_g}(\theta_i(N_{g,i})); \tag{6.8}$$

$$t_{max} = k_{r,c}n_r + k_{r,p}n_p + k_l \sum_{i=1}^{n_p} n_{l,i} + \sum_{i=1}^{n_g} \theta_i(N_{g,i}); \tag{6.9}$$

$$t_{avg} = k_{r,c}\sum_{i=1}^{n_r} p_{r,i} + \sum_{i=1}^{n_p}((k_{r,p} + k_l + n_{l,i})p_{p,i}) + \sum_{i=1}^{n_g} p_{t,i}\theta_i(N_{g,i}); \tag{6.10}$$

A detection system reaches its maximum scalability if majority of its processed packets are mapped to the best case (*i.e.,* t_{min}) while it becomes not practical

[2]For instance, if the ith telemetry is built on binary search tree, $\theta_i(N_{g,i})$ can be rewritten as $O(log(N_{g,i}))$ or $O(N_{g,i})$ for the average and worst case, respectively.

if most of packets have their processing time as the worst case (*i.e.,* t_{max}). To achieve a system's optimal practicality by converging its t_{avg} to t_{min}, I can make the following recommendations. First, minimizing the number of active parallel modules for a received packet in each step (*i.e.,* reducing n_r, n_p, and n_g in Eq. 6.10). Second, reducing probabilities of packet hit on costly modules (*i.e.,* $p_{r,i}$, $p_{p,i}$, and $p_{t,i}$). Third, selecting stateful telemetry that have light time complexity (*i.e.,* $\theta_i()$) and maintaining small number of entries ($N_{g,i}$) in the high-cost data structures to reduce the time consumption introduced by the "information gathering" step.

On the other side, an effective detection that provides precise inference of distributed attackers and malicious flows requires complex data structures to be involved, which often come with high cost. Recalling the walking example discussed in §6.3.1, the practical approach only detects victims with simple telemetry (light $\theta_i()$) of only protected enterprise hosts, whereas the effective approach detect victims, distributed sources, and malicious flows through a complex flow graph (heavy $\theta_i()$) containing all flow details between hosts. As benchmarked by my evalution (in §6.5.3) using real enterprise traffic, the effective detection has its per-packet time consumption $t_{effec.}$ about 21 times larger than the per-packet time consumption $t_{prac.}$ of the practical method.

Given the above considerations, a detection system is suggested to have complex data structures for effective detection, which are only used to process the necessary fraction of traffic, while the majority of packet streams are expected to be handled by light modules, so that the hit probability and complexity of costly modules will be significantly reduced for practicality. Next, I show my design of a multi-stage inference architecture based on this idea that detects distributed attacks progressively, whereby only a small but necessary fraction of packet streams are processed by high-cost modules and the majority of traffic are handled by low-cost processes.

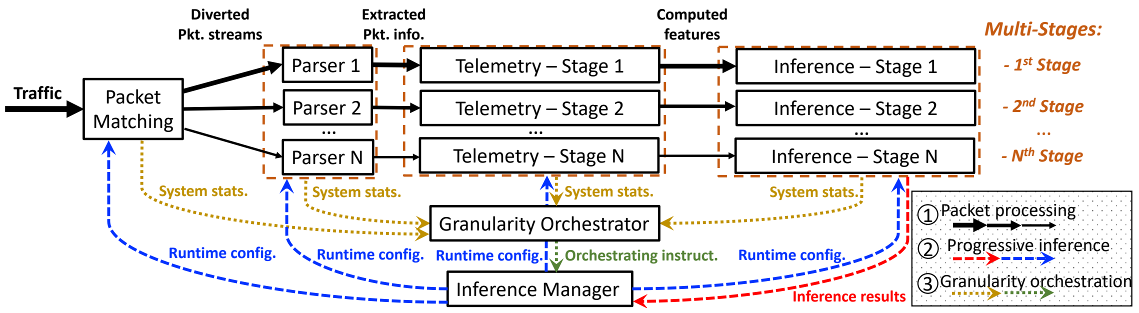


Figure 6.3: Design of PEDDA architecture – my multi-stage progressive inference method.

6.4 The Multi-stage Progressive Inference Architecture

In this section, I present my method as PEDDA architecture (§6.4.1) that employs multiple low-cost and high-cost inference stages to detect a distributed attack progressively, so that the only necessary fraction of traffic partitioned by an upper (high-cost) stage would be processed by its lower stage – they are dynamically instructed by reactive controls of programmable networks (§6.4.2). Given the uncertainty of complex traffic compositions, high-cost stages may still get overwhelmed if the finest granularity (IP-level) of telemetry is maintained during the progressive detection. To avoid this problem, I design an orchestrator that selects granularity (IP-level or coarse-grained subnet-level) of each stage by solving a run-time optimization problem bounded by available computing resources and current traffic compositions (§6.4.3).

6.4.1 Design of the PEDDA Architecture

I now describe the design of my multi-stage progressive inference architecture, including my design rationale and choices, schematic of the architecture, and its workflows.

Rationale and Choices

As revealed in Fig. 6.1(a), effective detection of a distributed attack requires a large scale monitoring of all aspects (*i.e.,* victims in the enterprise network, distributed external attackers, and malicious flows), which is not practical as the sizes of external attackers with flows are massive and unbounded. Therefore, legacy solutions either focus on only victims for practicality, or detect distributed sources and flows with high processing cost thus do not scale. I note that this trade-off could be resolved through multiple inference stages each makes a progression, so that telemetry is only maintained for the necessary fraction of traffic at each stage – an early stage monitors a simple aspect of the network (*e.g.,* enterprise hosts in Fig. 6.1(a)) through a low-cost telemetry, while the following stage (with a higher-cost telemetry) processes only a fraction of traffic partitioned based on the inference results from its prior stage. This process is recursively executed till an attack is fully detected at all aspects (*e.g.,* victims, sources, and flows).

Three design choices are made accordingly. First, I use multiple inference stages each achieves a progression (*e.g.,* victims, sources, or flows) in attack detection. They are dependent to each other and have logical orders. Second, an inference stage only processes the packet streams partitioned by its prior stage. Therefore, the first low-cost stage processes all received packet streams by default, whereas the stages with higher costs inspect only a minor fraction of traffic isolated by their prior cohorts. Third, each stage is expected to make its inference at the finest granularity of IP level. However, given the complexity of traffic and limited available computing resources, the granularity of each stage may get sacrificed (by aggregating into subnet level) to guarantee the operational robustness.

Schematic

Now I describe the schematic of my architecture as shown in Fig. 6.3. Driven by my design choices, my architecture has a number of dependent inference stages (a serial of packet matching, parser, telemetry, and inference models) placed by their logical orders with costs from low to high. An inference manager is used to collect detection results from all stages and reactively instructs them to expand or reduce the scope/fraction of traffic they process. To guarantee the operational robustness with limited computing resources, a granularity orchestrator is designed to digest run-time system statistics and adjust granularity of each stage accordingly – its specifications and mechanism will be discussed in §6.4.3.

Workflows

As shown in Fig. 6.3, there are three types of workflow in the PEDDA architecture, namely packet processing, progressive inference, and granularity orchestration.

Packet processing and progressive inference: During operations, the packet matching module receives traffic and proactively forwards all packet streams to the first stage which use the simplest (low-cost) telemetry. Therefore, it can handle all packet streams without any performance pressure. As an example, let's assume that the first stage detects only victims by tracking packet count of each enterprise host. Once a victim is identified, the first stage will send this result to the inference manager. The second stage will then be instructed to start processing traffic associated with this victim for a further progression (*e.g.,* detect distributed attackers). This procedure is recursively executed till the last stage to effectively detect all involved entities (*e.g.,* victims, attackers, and flows) in a distributed attack. With this progressive inference mechanism, only the necessary fraction of traffic is gradually processed by high-cost stages, which effectively detects attacks while ensuring the operational practicality.

Granularity orchestration: I note that the proposed architecture may still not be capable of process traffic with limited available computing resources under extreme scenarios (*e.g.,* most of the packet streams are malicious and require high-cost inspection). To maximize the robustness of my design, run-time system statistics of each pipeline such as traffic rate, CPU consumption, memory usage, and number of monitored entities are reported to the granularity orchestrator. The orchestrator determines whether to reduce granularity (*i.e.,* from IP level to coarse-grained subnet levels) of each stage by solving an optimization problem (in §6.4.3). Orchestrating instructions are sent to the inference manager and then reflected on each traffic processing stage through run-time configurations.

6.4.2 Choices of Reactive Control

The key enabler of my progressive inference method is reactive control of packet forwarding and processing, which is naturally matched with the emerging paradigm of programmable networks (*i.e.,* SDN and NFV). In what follows, I discuss three possible choices to achieve the reactive control of my proposed architecture using various technologies of programmable networks.

First, the presented architecture can be realized with only virtual network functions (VNF). The packet matching module and packet parser bank in Fig. 6.3 can be operated within a software switch (*e.g.,* vSwtich) that interact with other components configured as modular services on generic servers. This choice is practical for small enterprises with low traffic rates and may not have hardware SDN switches in operation. However, purely software-based packet processing is not practical for handling high-throughput traffic of large enterprises [10]. Second, to be suitable for high-rate environments, my architecture can be realized with programmable control plane hardware switches (*e.g.,* OpenFlow) as its packet processing modules, which can handle packet streams at line rate. Other components including stateful telemetry are still operated as software services. Third, one may want to offload simple telemetry to programmable data plane switches (*e.g.,* P4) to save its computing resources on generic servers. However, this technology is still in its early stage of industrial adoption, which makes it not very practical for current enterprises.

6.4.3 Orchestrating Stage Granularity

Intuitively, my PEDDA architecture achieves effective detection while satisfying the requirement of practicality through progressive inference of multiple stages described in §6.4.1. However, stages especially for those with high-cost telemetry may still get overwhelmed by extreme traffic scenarios. For example, if the majority of traffic are malicious and from well distributed sources with massive number of concurrent flows, they may fail to monitor such complexity of sources and flows at the finest granularity of IP level.

To guarantee the operational robustness of my PEDDA, I develop an orchestrator (already shown in Fig. 6.3) that dynamically selects granularity (*i.e.,* at the finest IP level or coarse-grained subnet levels) of each stage to maintain the size of telemetry at an acceptable range given the current traffic complexity and available computing resources. Ideally, all inference stages have their finest granularity at IP-level (*i.e.,* subnet mask as "`/32`"), that is, statistics are maintained for each IP address. However, if the available computing resources are not enough to support IP-level monitoring, granularity (*i.e.,* subnet mask) of the less important stages defined by users are reduced to free up more resources. For the worst case, a less important stage would have its subnet mask as "`/0`" – it monitors the entire network as one entity (*i.e.,* `0.0.0.0/0`), so that no additional cost will be introduced even if the complexity of network is high.

Given the above logic, my orchestration approach can be mathematically formulated as a constrained optimization problem described as follows.

The Objective Function

The objective of my orchestration is to have better granularity (*i.e.,* larger subnet mask for the monitored network entities) for stages with higher priorities. Assuming that there are N_s stages, and the ith stage has its subnet mask as S_i and priority as W_i, my objective function can be written as formula.6.11.

$$\max \sum_{i=1}^{N_s} (W_i \cdot S_i) \tag{6.11}$$

Constraints

I now articulate the constraints of my orchestration for subnet mask range, switch resources, and server utilization.

Subnet mask range: Since an IPv4 address has 32 binary digits, the subnet masks of all stages fall into a fixed range from 0 to 32 as specified in formula.6.12. For an IPv6 address, the upper bound range is 128 digits which is not considered in this chapter. Network administrators can also use a customized range within the interval.

$$\forall i \in [1, N_s] : 0 \leq S_i \leq 32 \tag{6.12}$$

Switch resources: The "packet matching" module that receives packet streams and diverts them into each inference stage has its flow rules changed at run-time by reactive configurations. This module is realized by software of hardware programmable switches that have limited rule tables. A flow rule could be created for a single IP address or an aggregated subnet defined by its mask. Therefore, I use H_i to denote the current number of IP addresses to be tracked by the ith stage, given the current subnet mask S_i, the number of flow rules required by the stage can be estimated as $\frac{H_i}{\gamma(S_i)}$, where $\gamma()$ is an approximated bounded coefficient. Ideally, $\gamma()$ represents the perfect subnet aggregation, *i.e.,* a flow rule with its subnet mask S_i covers 2^{S_i} active IP addresses; while in the worst case, all active IP addresses are sparsely distributed and one flow rule can only match one host, *i.e.,* $\gamma(S_i) = 1$. Given the maximum number of flow rules F_i and changing rates δF_i, I have constraints for the two metrics shown in formula.6.13.

$$\forall i \in [1, N_s] : \frac{H_i}{\gamma(S_i)} < F_i; \forall i \in [1, N_s] : \delta \frac{H_i}{\gamma(S_i)} < \delta F_i; \tag{6.13}$$

If a programmable data plane switch (*e.g.,* P4) is used as the "packet matching" module, two additional constraints are introduced for each stage including maximum SRAM R_i and stateful ALU A_i allocated to the ith stage. I use r_i and a_i to represent the SRAM and stateful ALU usage per monitored entity, and the two constraints are expressed as in formula.6.14.

$$\forall i \in [1, N_s] : r_i \cdot \frac{H_i}{\gamma(S_i)} < R_i; \forall i \in [1, N_s] : a_i \cdot \frac{H_i}{\gamma(S_i)} < A_i; \tag{6.14}$$

Server utilization: The performance of modules operated on commodity servers are bounded by its allocated CPU and memory utilization. I use $C_{parser,i}$, $C_{telemetry,i}$, $C_{inference,i}$ to represent the maximum CPU utilization allocated to the parser, telemetry, and inference of the ith stage, respectively. Assuming that the packet arrival rate on the ith stage is λ_i, and the inference frequency is f_i, per packet CPU time consumption for each step can then be represented as $c_{parser,i} \cdot \lambda_i$, $\lambda_i \cdot \theta_{telemetry,i}(\frac{H_i}{\gamma(S_i)})$, and $c_{inference,i} \cdot \delta \frac{H_i}{\gamma(S_i)} \cdot f_i$, where $c_{parser,i}$ and $c_{inference,i}$ are constant coefficients and $\theta_{telemetry,i}()$ is the estimated time complexity function of the stateful telemetry. Constraints for CPU utilization of packet parser, telemetry, and inference are expressed as in formula.6.15,6.16, and 6.17, respectively.

$$\forall i \in [1, N_s] : c_{parser,i} \cdot \lambda_i < C_{parser,i} \tag{6.15}$$

$$\forall i \in [1, N_s] : \lambda_i \cdot \theta_{telemetry,i}(\frac{H_i}{\gamma(S_i)}) < C_{telemetry,i} \tag{6.16}$$

$$\forall i \in [1, N_s] : c_{inference,i} \cdot \frac{H_i}{\gamma(S_i)} \cdot f_i < C_{inference,i} \tag{6.17}$$

Both parser and inference module are stateless functions, which consume a roughly constant and negligible amount of server memory. As for stateful telemetry, I use $m_{telemetry,i}$ to denote the run-time memory usage per monitored entity of the ith stage. Therefore, the constraint for memory consumption by telemetry can be expressed as in formula.6.18.

$$\forall i \in [1, N_s] : m_{telemetry,i} \cdot \frac{H_i}{\gamma(S_i)} < M_{telemetry,i} \tag{6.18}$$

During operations, network administrators of an enterprise specify the priority, complexity function of telemetry, CPU and memory constant coefficients and limitations of each stage that could be estimated from empirical benchmarking. Later in §6.5.3, I will demonstrate how the run-time optimization orchestrates granularity of each stage to achieve system robustness under constrained resources.

6.5 Realizing and Evaluating A Proof-of-Concept Prototype

In this section, I present my proof-of-concept prototype of the *PEDDA* architecture. It uses three practical inference stages driven by the insights from an empirical traffic analysis of a large representative enterprise (§6.5.1). The prototype implementation details are discussed in §6.5.2. Through evaluations using one-day worth of enterprise traffic and ground-truth DDoS attacks, I demonstrate that my prototype outperforms its counterparts in telemetry visibility, detection effectiveness, and operational robustness (§6.5.3).

6.5.1 Three Practical Inference Stages

Choosing suitable inference stages for enterprise networks is an important task to realize the PEDDA architecture (discussed in §6.4) as a practical system. To motivate my selection of inference stages, I perform an empirical analysis on traffic traces captured from the network edge of a large enterprise during a one-day period to understand its flow and host profiles. Driven by the insights, I design three practical inference stages with different telemetry and detection progression that identify active enterprise hosts, victims, distributed sources with flows of volumetric attacks, respectively.

Analysis of Traffic from an Enterprise

In my conceptual design, the majority of packet streams are processed by low-cost stages for early inference while a minority of traffic are reactively inspected by high-cost stages. To this end, understanding the traffic profile of a typical enterprise is the prerequisite step to choose a practical and effective approach of progression. In what follows, I discuss my empirical analysis of flow and host profiles in an enterprise network and the obtained insights that motivate my stage design.

Dataset: I provisioned a full mirror of inbound and outbound traffic from the network edge of a large-sized university through two 10 Gbps fiber links separately[3]. I collected the first 96 bytes of packets for one working day (31 May 2019 12:00 PM to 1 June 2019 12:00 PM) with negligible dropping rates (*i.e.,* less than 0.05%), resulting in 13.8B inbound and 21.5B outbound packets. During the busy hours from

[3]UNSW Human Research Ethics Advisory Panel approval number HC17499, and CSIRO Data61 Ethics approval number 115/17.

13:00 to 16:00, the inbound link carried about 9 Gbps traffic with a peak packet rate as 700K pps, and the outbound link had its throughput as about 2 Gbps with 600K pps.

Flow profiles: I now analyze the profile of flows crossing the enterprise. The direction of a flow (*i.e.,* inbound or outbound) is defined by its first packet. In the entire dataset, I observe 60.8M outbound flows and a much larger number (559.2M) of inbound flows. Surprisingly, the majority of them (*i.e.,* 72.6% for outbound and 68.8% for inbound flows) only have packets for one direction. By investigating into those abnormal flows, I found out that 36.4% of single directional flows only have 1 packet, while the rest of them are mostly with repetitive packets – they are likely to be linked with network scans and mis-configurations that correspond to a small fraction of packets (4.26% for outbound and 5.57% for inbound).

Host profiles: There are a total of 304K enterprise IP addresses that appeared in my dataset during the day. I first analyze their packet distribution. The majority (92.4%) of enterprise IPs only have inbound packets without sending outbound traffic – they are either unassigned IP addresses of the enterprise or inactive hosts. On the contrary, I do not observe any enterprise IP address that only sends outbound packets without receiving inbound traffic.

Next, I consider the flow distribution among each enterprise host. Unsurprisingly, the majority (84.5%) of inbound single directional flows target unassigned IPs or inactive enterprise devices, as they are likely to be network scans that randomly select their target. As for outbound single direction flows, they are all densely generated by a negligible fraction (0.4%) of enterprise hosts that exhibit abnormal behaviors such as performing port scans toward hosts on the Internet.

Considering the statistics of external hosts, during the day, 751K external hosts sent packet towards enterprise IPs while only 59% of them get replied. Besides, I also observe a small amount (15K) of external IP addresses that only received packets from enterprise hosts.

Highlights: I now recap two key insights that inspire the design of my practical inference stages. First, a small amount of unsolicited packets targeting inactive enterprise and external IP addresses create a large number of single directional flows across the network, which is particularly true for the inbound direction. Therefore, by only focusing on active hosts that need protection against distributed attacks, the complexity and size of high-cost telemetry could be significantly reduced. Second, compared with enterprise hosts, external hosts and concurrent flows are massive in amount (up to 750K and 600M, respectively). Maintaining statistics of external hosts and flows for all active enterprise hosts becomes impractical. Thus, it is necessary to limit the number of monitored external hosts and flows through low-cost stages.

Specification of the Three Stages

Now I discuss the specification of three practical stages designed for an enterprise network that detect active enterprise hosts, attack victims, and distributed attackers with malicious flows, respectively.

The first stage: My first stage is designed to detect *active enterprise hosts*. The packet matching module sends all outbound packet streams to this stage by default. The parser of this stage extracts source IP addresses of outbound packets, which are used to update a list of active enterprise IP addresses (or subnets if its granularity is reduced). An IP address will get removed from this telemetry if it has no outbound packet for a user-defined period. Changes of the list are reported to the inference manager periodically, so that corresponding reactive configurations will be generated for the subsequent stages.

The second stage: My second practical stage identifies enterprise hosts that are *victims* in distributed attacks. As reactively instructed by the inference manager, it receives inbound and outbound traffic of only active enterprise hosts identified by the first stage. The telemetry maintained for this stage monitors traffic statistics of each active enterprise host to detect victims from benign ones. In my proof-of-concept prototype, by following the practice of state-of-the-art detection system [202], I track packet arrival rates of both inbound and outbound directions for each monitored entity (IP or subnet). If the difference between inbound and outbound packet rates (*i.e.,* $\#Pkt.In - \#Pkt.Out$) of a monitored entity exceeds a user-defined threshold, the inference module will report it as a victim, so that the inference manager can instruct the following stage accordingly.

The third stage: My third practical stage maintains high-cost telemetry to detect *distributed attackers* and *malicious flows* in a network attack. This high-cost stage only receives traffic of identified victims. For each packet, the parser of this stage extracts its size, protocol, source and destination IP addresses and port numbers. A streaming graph is used as the telemetry that tracks each network flows between external hosts and the victims. Attributes of each external host and flow are computed periodically, so that external sources and malicious flows can be precisely detected at this final stage. In my prototype, I use the number of active flows between an external host and the victim as my detection criteria, which is commonly used by practical security middleboxes [204, 294]. I note that the inference function could be further improved by more descriptive attributes and precise algorithms [5], which is not in the focus of this paper.

6.5.2 Implementation Details

To realize my PEDDA architecture with the three practical stages, I implement a proof-of-concept prototype using an OpenFlow programmable switch and software modules operated on a commodity server. Fig. 6.4 shows its functional blocks and their interactions. Inbound and outbound traffic entering and leaving the enterprise

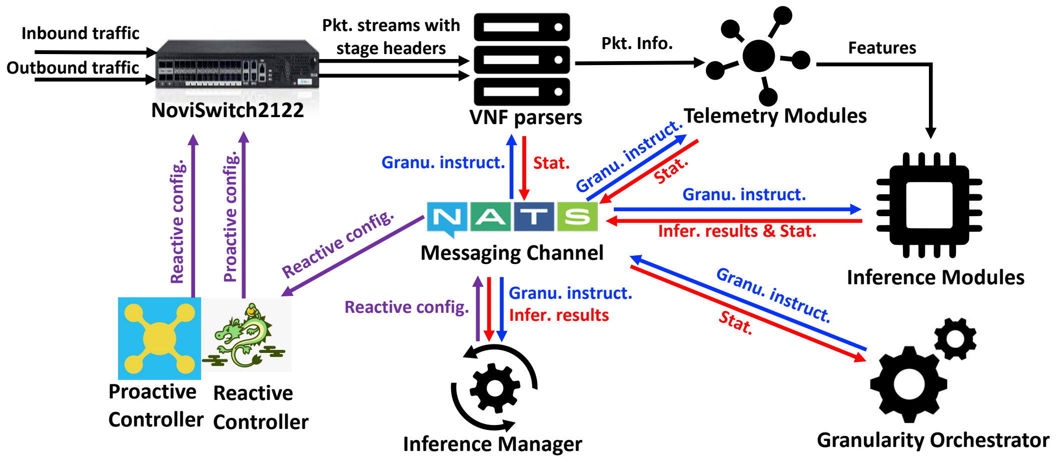


Figure 6.4: Proof-of-Concept Prototype Implementation.

are mirrored to two 10 Gbps network interfaces of the OpenFlow switch (NoviFlow 2122). The programmable switch annotates each received packet by adding an extra header indicating the stage it goes. VNF parsers (using DPDK framework and NFF-Go library), telemetry, and inference modules of my three practical stages are written in Golang and deployed on a blade server with sixteen 2.10GHz CPUs and 64GB RAM. A publish-subscribe messaging channel (NATS) is used to exchange inference results, reactive configurations, system statistics, and orchestrating instructions. The granularity orchestrator (written in Golang) updates the subnet masks of each stage by solving optimization problems using the received system statistics as discussed in §6.4.3. The inference manager (written in Python3) publishes run-time configurations for the three inference stages based on detection results and granularity orchestrating instructions received from the messaging channel. Last, there are two SDN controllers (Faucet and Ryu) that send proactive and run-time reactive configurations to the switch, respectively.

6.5.3 Evaluation and Comparison

To evaluate the efficacy of my prototype, I first describe a demonstrative distributed attack detection example by my prototype with three-stage progressive inference; I then experimentally compare my prototype with other state-of-the-art solutions to highlight its merits in three aspects including telemetry visibility, detection accuracy and responsiveness, and operational robustness.

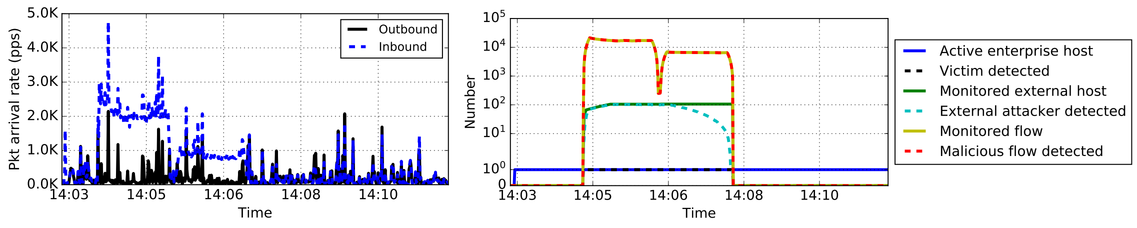
Demonstrative Attack Detection

I demonstrate the detection workflow of my prototype by replaying a snippet dataset of real enterprise traffic injected with ground-truth DDoS attacks.

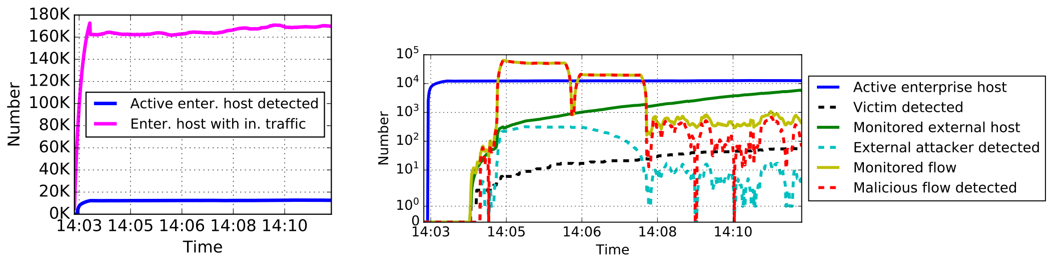
Dataset: To create my demonstration dataset, I first select a publicly available traffic trace file containing 10 minutes DDoS attacks [296] as the ground truth, and a 15-minute snippet of my campus trace file (discussed in §6.5.1) as the background traffic. I load the ground-truth DDoS attacks on three representative servers (*i.e.,* a website server, a VPN gateway, and a student portal) by changing their destination IP addresses. As the result, I obtained a 15-minute dataset containing well-labeled distributed network attacks on three enterprise servers.

Detection thresholds: The detection thresholds of my three practical stages are configured as suggested by common practices [203, 204, 294] of the security community. For the first stage, an active enterprise host will get removed if no outbound packet is sent for 10 seconds. For the second stage, if the difference between inbound and outbound packet rates of a monitored host exceeds 500 pkt/s, it will be reported as victim. For the third stage, if an external host contacts the victim with more than 10 active connections, the host and its flows will be flagged as anomalies.

Results and statistics: The evaluation dataset is replayed into my proof-of-concept prototype. The run-time detection statistics are shown in Fig. 6.5, which demonstrate that my three practical stages are effective in identifying active enterprise hosts, victims, distributed sources, and malicious flows by only monitoring the necessary entities.



(a) Traffic rate – a web server. (b) Detection statistics of the web server.



(c) The 1st stage. (d) Detection statistics of the entire dataset.

Figure 6.5: Demonstrative detection statistics by my proof-of-concept prototype with three inference stages.

I first take the website server as a case study. As illustrated in Fig. 6.5(a), the inbound and outbound packet rate of this server is roughly equal and below 1K packets per second (pps) during the most of time, while an exception happened from 14:04 to 14:08 when I load an attack with about 10K concurrent flows from 100 distributed sources. As visually shown in Fig. 6.5(b), my three inference stages effectively detect all distributed sources and their flows. On the first stage, the server was consistently labeled as an active host for its outbound activity (shown by the black line). The second stage tracked its packet rates for both directions and detected it as a victim (blue line) during the attack period. Inbound and outbound packet streams of the victim were reactively processed by the third stage to identify external sources and malicious flows. As shown in Fig. 6.5(b), the majority of external hosts and flows contacting the victim were labeled as anomalies, whereas a small number (less than 5, thus looks negligible in the figure) of external hosts still communicated with the server.

Now I report the overall detection statistics on the entire dataset. As shown

Table 6.1: The overview statistics of my experimental evaluation for PEDDA and other state-of-the-art solutions.

	Telemetry Visibility				**Detection Accuracy**			**Operational Cost**	
	Host	**Victim**	**Attacker**	**Flow**	**Victim**	**Attacker**	**Flow**	**CPU (Avg./Peak)**	**RAM (Avg./Peak)**
NGFW	Partial	Partial	None	None	100%	0%	0%	20%/42%	not measurable
IDS	Partial	Partial	None	None	100%	0%	0%	35%/62%	21%/34%
Flow Graph	Complete	Complete	Complete	Complete	100%	95.4%	94.1%	>700%/Full	Full/Full
PEDDA	Complete	Complete	Complete	Complete	100%	93.4%	91.7%	24%/57%	8%/11%

in Fig. 6.5(c), a large amount (170K) of IP addresses in my university IP blocks were targeted by inbound packets during run-time, whereas only about 10% (18K) of them are active hosts identified by the first practical stage. As instructed by my progressive inference mechanism, traffic of only active enterprise hosts were processed by the second stage, which detects victims in a distributed attack. The blue line in Fig. 6.5(d) shows the number of potential victims detected by the second stage. All 3 victims in my ground-truth attacks are successfully detected by the second stage. In addition, there are 13 other enterprise hosts identified as victims. As the consequence, inbound and outbound traffic of those detected victims are reactively processed by the third stage. The number of external hosts and flows that are processed and detected as anomalies by this stage is shown in Fig. 6.5(d). By comparing with my ground-truth records, all external sources and their malicious flows are detected by the stage. On average, about 1K external hosts are monitored at run-time, and only about 1% of them are actual attackers (shown as the blue lines). Furthermore, as shown by the yellow and red lines in Fig. 6.5(d), the majority ($\geq$90%) of monitored flows are identified as malicious, which is understandable as external attackers tend to overload a victim by massive amount of flows, while benign users do not hold many concurrent connections.

Noting that there are around 260K external hosts and 600K concurrent flows in my dataset. Through the progressive inference of my three stages, less than 4K external hosts and 1K flows are tracked by the high-cost stage for fine-grained detection, which achieves both practicality and detection effectiveness.

Experimental Comparisons

I experimentally compare my prototype with two widely adopted solutions (*i.e.,* a NGFW as Palo-Alto firewall appliance PA-3020 [202] and an IDS as Zeek [97] network security monitoring tool of version 3.1.0-dev.280) and one flow graph that is identical to the one used in my third stage. The NGFW is operated as a standalone proprietary hardware appliance while the other systems (my prototype, IDS, and the flow graph) are deployed on the blade server that is already described in §6.5.2. All systems are configured using the same detection thresholds so that the inference architecture is the only variable in my comparisons. During my experimental evaluations, my ground-truth DDoS attack dataset (discussed in §6.5.3) was replayed continuously together with the enterprise background traffic of a day. In total, I obtained 432 DDoS attacks on enterprise servers from 43.2K attackers and more than 4.32M malicious flows.

Telemetry visibility: The telemetry visibility of each solution into enterprise hosts, victims, attackers, and malicious flows is reported in the respective region of Table 6.1. The NGFW often provides telemetry only for key enterprise servers. IT departments have to manually specify the IP addresses or subnets [297] that require monitoring. Due to the scalability concern, it does not provide visibility into external hosts and flows for DDoS detection. Therefore, such systems can handle a large volume of traffic but also cause collateral damage when mitigating attacks [9]. The software IDS requires IT department to install scripts that specify its detection logic and matching signatures. It provides flexibility to users who want to have their customized telemetry for detection. However, such systems are operated on generic CPUs that cannot handle large throughput traffic for complex logic [10]. Thus, only monitoring enterprise hosts that are important is recommended by the community [298]. My prototype provides a more flexible and fine-grained telemetry visibility than its counterparts. It monitors all active enterprise hosts without specifications from network administrator, and reactively maintains visibility into external hosts

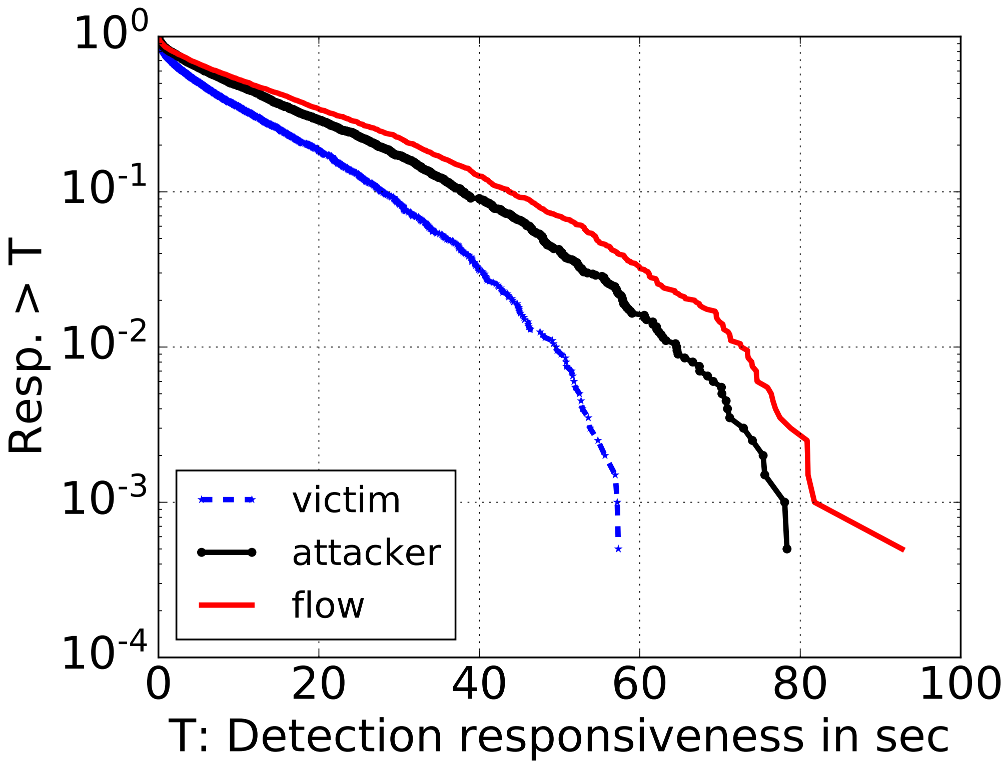
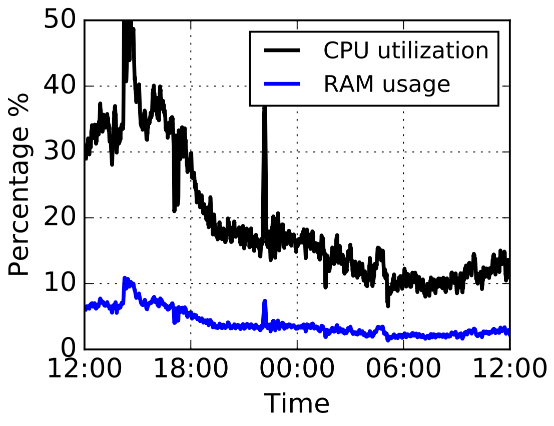


Figure 6.6: Detection responsiveness of my prototype.

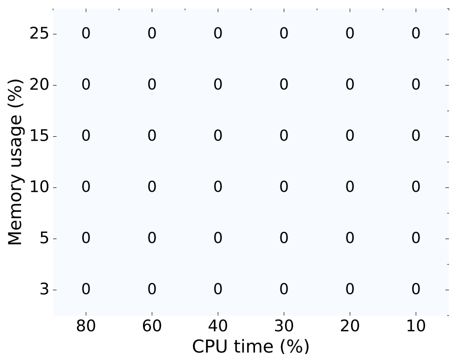
and flows that contact victims in a distributed network attack. Such visibility can only be achieved by the flow graph which is too computationally expensive for real-time operation.

Detection accuracy & responsiveness: I now report the detection accuracy and responsiveness for victims, external attackers, and malicious flows. As for detection accuracy, according to Table 6.1, my approach achieved satisfactory detection results for victims, sources, and malicious flows (100%, 93.4%, and 91.7% respectively). Whereas typical enterprise solutions that are mounted with DDoS detection rules and signature scripts (*i.e.,* NGFW and IDS) can raise alarms for all victims (with 100% accuracy) but are not able to differentiate distributed attackers and malicious flows (resulting in 0% accuracy). Fine-grained detection using flow graphs achieved the best results (100%, 95.4%, and 94.1% for victim, attacker, and flow respectively). It is able to detect distributed sources and malicious flows that are missed by the progressive inference of my prototype. However, maintaining more than a half million flow records renders it unable to operate in real-time (*i.e.,* taking more than 7 days to process my 1-day traffic). I also report the responsiveness of my system in detecting victims, sources, and malicious flows as shown in Fig. 6.6. The majority of attacks can be detected within 20 seconds at victim- (81.6%), source- (71.1%), and flow-level (66.0%), and almost all attacks are detected within a minute at all three levels (100%, 98.4%, and 96.8%), respectively. I observe that some enterprise victims sent more outbound packets than inbound ones during some periods, thus, they are not able to be responsively detected through my thresholds on traffic rates. As the result, a small fraction of external attackers (6.4%) and malicious flows (8.3%) are missed. I note that a more precise behavioral profiling of attacker and flow could be used to address the problem, which is not in the scope of this chapter.

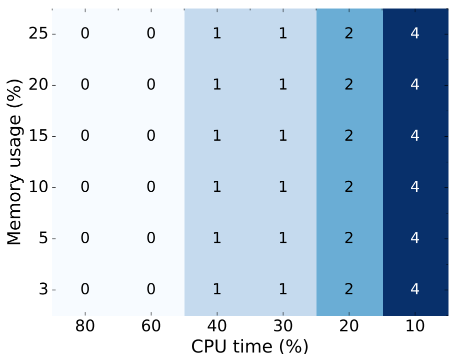
Operational robustness: As shown in the rightmost region of Table 6.1, compared with its counterparts, PEDDA has a reasonably light consumption on computational resources (*i.e.,* CPU and RAM) while it has better (or equivalent) performance in the other two aspects. I note that the flow graph consumed more than 7 days for a 1-day traffic traces, therefore, its average CPU usage is marked as "$>$700%". To guarantee the operational robustness regardless of traffic compositions and resource availability, my system uses a granularity orchestrator (described in §6.4.3) that adjusts subnet masks of each inference stage to regulate its resource consumption. I now demonstrate its usefulness. My prototype did not face shortage of computing resources when handling real campus traffic with peak rate of 10 Gbps, as evidenced by the CPU and RAM usage shown in Fig. 6.7. Therefore, all three stages maintain the finest granularity of IP level with their subnet mask as `/32`. To emulate a less powerful platform, the maximum CPU utilization of the dedicated cores and RAM usage are set from 10% to 80% and 3% to 25% gradually. My prototype was operated without compromising granularity (*i.e.,* no reduction on subnet mask) when it was assigned with more than 15% memory and 60% CPU resources. With stricter limitations applied, the first stage that tracks active enterprise hosts did not have any reduction on its subnet mask (in Fig. 6.7(b)) under all



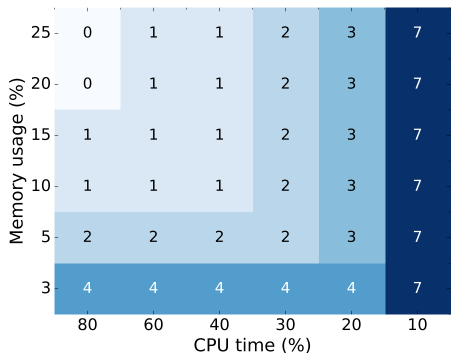
(a) Server resource usage.



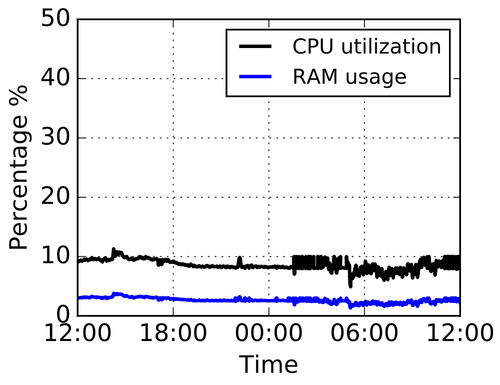
(b) Mask reduction: 1st stage.



(c) Mask reduction: 2nd stage.



(d) Mask reduction: 3rd stage.



(e) Resource usage with constraints.

Figure 6.7: Operational robustness: (a). server CPU and RAM usage without resource constraints; subnet mask reductions of the (b). 1st, (c). 2nd, (d). 3rd stages during peak traffic rate under various resource constraints; and (e). resource usage with the constraints of 10% CPU and 3% RAM.

circumstances, as it is granted the highest priority during orchestrations, whereas the second stage (in Fig. 6.7(c)) that detects enterprise victims and the third stage

(Fig. 6.7(d)) that identifies external sources with malicious flows have their subnet masks reduced up to 4 and 7 digits, respectively. In Fig. 6.7(e), I show the real-time CPU and RAM consumption of my prototype when only 10% CPU and 3% RAM are available. It is clear that my orchestration effectively controls the run-time resource consumption below the defined limits. As a summary, the robustness of my prototype is guaranteed through the dynamic granularity orchestration for each stage given the available computing resources.

6.6 Conclusion

Current distributed attack detection solutions for enterprise networks are either not effective in identifying sources and malicious flows to avoid collateral damage, or not practical to be operated under the high-throughput network environment of an enterprise. In this chapter, I proposed PEDDA, a progressive inference method that achieves both effective detection and operational practicality via multiple stages orchestrated by the dynamic control of programmable networks. First, I highlight the performance bottlenecks of traffic processing in legacy detection solutions by mathematical formulation of its time complexity. Second, I design my method as PEDDA architecture that detects distributed attacks progressively through multiple inference stages, each with a certain cost and orchestratable granularity. During operations, packet streams are progressively partitioned and processed by the stages with increasing costs, the granularity of which is orchestrated through optimization constrained by available computing resources. Third, I implement and evaluate a proof-of-concept prototype that uses three practical inference stages to detect active enterprise hosts, victims, sources with flows progressively. Evaluation results show that PEDDA outperforms legacy solutions and is ready for deployment at an enterprise scale.

Chapter 7

Conclusions and Future Work

Contents

7.1 Conclusions

Enterprise networks are frequently targeted by distributed network attacks of diverse types and patterns. To cope with such ever-increasing cyber threats, enterprise IT departments are expected to have a seamless perception toward the profile of connected assets within their networks and provide well-tailored detection against network attacks on vulnerable assets accordingly. Existing solutions for asset behavioral tracking and distributed attack detection are either static thus struggle to cope with the fast-changing behavioral profiles of enterprise hosts and external attackers, or via costly statistical methods using fine-grained telemetry that are not practical to be maintained for a large enterprise network.

In this thesis, by leveraging data-driven techniques and programmable network telemetry, I developed a series of methods and practical prototypes for enterprise

IT departments to address the above security “blind spot” from DNS to the whole dimension of network traffic across an enterprise edge, covering asset behavioral classification and distributed network attack detection.

The key contributions presented in this thesis toward enterprise network security are briefly summarized below.

- Towards DNS asset profiling and classification, I developed an automatic method to classify DNS assets using cluster algorithms and track their cyber-health through a set of well-defined metrics. I analyzed over 1 million DNS packets from two large organizations to profile the DNS behavior of enterprise hosts. I identified key attributes and develop a method using clustering algorithms to classify DNS assets into authoritative name servers, recursive resolvers, mixed DNS servers, and end-hosts behind or not behind the NAT. For those identified DNS assets, I proposed a set of monitoring metrics to track their DNS traffic health that helps us to identify various types of DNS anomalies. My method was applied to 32-day DNS traces and reveals unknown knowledge for the respective IT departments to address their “DNS blind spots” such as DNS servers configured by affiliations and vulnerability exploited by external attackers that could be fixed through secure configurations.
- Towards asset network behavioral monitoring and classification, I designed and prototyped a real-time SDN system using a multi-grained classification scheme to give fine-grained functional types or coarse-grained transport-layer behavioral types of enterprise hosts. My system reactively collects packet-level telemetry of hosts potentially with anomalies for forensics analysis. I performed a comprehensive big data analysis on over 3 billion packets from a representative organization to highlight typical and non-typical behavioral types of enterprise hosts. Driven by the insight, I developed a multi-grained classification scheme using well-selected attributes and supervised ML models to classify enterprise hosts into either typical fine-grained types (*e.g.,* website

server and mail server) or non-typical coarse-grained types (*e.g.,* TCP server and UDP proxy). I then prototyped a real-time system that classifies enterprise hosts and reactively inspects packet-level telemetry of suspicious hosts through dynamic flow rules of programmable switches. My system is practically deployed at an enterprise network for over a month and identifies thousands of unknown assets and network anomalies of which the IT department are not aware.

- For the problem of distributed DNS attack detection, I proposed a hierarchical anomaly-based architecture that effectively detects external DNS attackers even at distributed and stealthy manner. I started by analyzing over 400 million DNS packets from two organizations to profile distributed DNS attacks including scans and DDoS. I then developed a hierarchical data structure to profile DNS volumetric queries from external entities at various levels of subnet aggregations and train/tune/evaluate anomaly detection models to detect external attackers at host-, subnet-, and AS-level. I then prototyped a real-time detection system and deployed it for over a month. Evaluation results show that it is able to detect DNS attacks even in a well-distributed and stealthy manner, which might be missed by the legacy solutions.

- As for the effective and practical detection of distributed network attacks on enterprise assets, I came up with the design and implementation of PEDDA system that detects distributed network attacks from light to expensive tasks through a multi-stage progressive inference architecture. I formally modeled the traffic processing process of state-of-the-art attack detection systems to identify their performance bottlenecks. I proposed the PEDDA architecture that uses a multi-stage progressive inference method to detect a distributed network attack from simple to complex stages gradually. The telemetry granularity of each stage is dynamically orchestrated through an optimization process. I then designed and built a proof-of-concept prototype for enterprise networks using a three-stage detection that identifies active enterprise hosts, isolates victims, and detects distributed attackers with malicious flows, respectively. PEDDA is evaluated using real enterprise traces and public DDoS datasets. The results demonstrate its ability in detecting attacks effectively until the flow level while practical for a large organization, which is hard to realize with current enterprise solutions.

7.2 Future Work

I believe that the contributions presented in this thesis provide significant references for effective and practical enterprise network security. It is worth noting that my methodologies, system designs, and prototypes could be further improved and extended as the natural follow-ups on this thesis. Several example future directions are outlined as follows.

- In Chapter 3, I tracked various types of unhealthy DNS traffic characteristics as an inference for identifying DNS anomaly types. Due to the limitation of my dataset that only contains two representative enterprises, I used a codebook method and developed mapping for a limited number of popular anomaly types. It could be further improved by expanding the coverage of anomaly types from a more inclusive dataset and developing precise inference methods through ML techniques.
- My real-time asset classification scheme presented in Chapter 4 gives fine-grained labels to those enterprise hosts with typical functionality types such as website servers and proxies, while the rest are assigned with their coarse-grained types (such as TCP servers and proxies) with notations of their dominant transport-layer services. I believe that the coarse-grained inference results could be further clustered and refined by their communication patterns

(*e.g.,* commonly used protocols and destination IP addresses) to provide more insightful references for asset security management.

- The PEDDA system described in Chapter 6 maintains network telemetry for distributed attack detection at various levels of dynamic stages such as active enterprise hosts, victims, distributed attackers, and malicious flows. I acknowledge that the attack detection at each stage inherits simple thresholds from legacy solutions, which could be further enhanced by using more accurate methods such as statistical learning models on a comprehensive set of features derived from network telemetries.

- My two contributions on DNS security (in Chapter 3 and 5) focused on unencrypted DNS which is predominately adopted by the current ecosystem. With the growing concerns on data privacy and advances in DNS encryption (*e.g.,* DNS-over-TLS [192] and DNS-over-HTTPS [193]), the networking community starts to accelerate its industrial adoption by addressing related practical challenges [299]. With encrypted DNS, key attributes computed from DNS query names will not be available to my inference systems. Therefore, refining my data-driven methods for compatibility with encrypted DNS is a valuable future direction.

- Due to the deployment location of my telemetry infrastructure, in this thesis, I only consider the network traffic across an enterprise edge (*i.e.,* through the border routers that separate internal networks and the public Internet). Thus, I could not investigate the communications between hosts within an enterprise network, which will inevitably lead to significant security findings such as internal malware spreading, infiltration attacks, and others. I believe that future explorations of network traffic within an enterprise supported by a telemetry infrastructure with better visibility would bring astounding knowledge to the community.

Apart from the future directions listed above, there are many other aspects of enterprise network security that are not covered by this thesis, I hope they could be explored by my community in the near future.

References

[1] M. Lyu, H. Habibi Gharakheili, and V. Sivaraman, “A Survey on Enterprise Network Security: Asset Behavioral Monitoring and Detection of Distributed Attacks”, 2021.

[2] M. Lyu, H. Habibi Gharakheili, C. Russell, and V. Sivaraman, “Mapping an Enterprise Network by Analyzing DNS Traffic”, in *Proc. Springer PAM*, Puerto Varas, Chile, Mar. 2019.

[3] M. Lyu, H. Habibi Gharakheili, and V. Sivaraman, “Analyzing Enterprise DNS Traffic to Classify Assets and Track Cyber-Health”, 2021.

[4] M. Lyu, H. Habibi Gharakheili, and V. Sivaraman, *Under review*, 2021.

[5] M. Lyu, H. Habibi Gharakheili, C. Russell, and V. Sivaraman, “Hierarchical Anomaly-Based Detection of Distributed DNS Attacks on Enterprise Networks”, *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 1031–1048, 2021.

[6] M. Lyu, H. Habibi Gharakheili, and V. Sivaraman, “PEDDA: Practical and Effective of Distributed Attacks on Enterprise Networks via Multi-stage Progressive Inference”, 2021.

[7] H. Solomon, *Rio Games Faced Olympic-Sized DDoS Attacks*, `https://www.itworldcanada.com/article/rio-olympics-faced-olympic-sized-ddos-attacks/386207`, Accessed: 2020-08-18, 2020.

[8] B. Sullivan, *Rio 2016 Olympics Suffered Sustained 540Gbps DDoS Attacks*, `https://www.silicon.co.uk/security/rio-olympics-ddos-attacks-196998?inf_by=5aee98d7671db877298b45c2`, Accessed: 2018-05-01, 2018.

[9] C. Dietzel, M. Wichtlhuber, G. Smaragdakis, and A. Feldmann, “Stellar: Network Attack Mitigation Using Advanced Blackholing”, in *Proc. ACM CoNEXT*, Heraklion, Greece, Dec. 2018.

[10] M. Zhang *et al.*, “Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches”, in *Proc. NDSS*, San Diego, CA, USA, Feb. 2020.

[11] S. K. Fayaz *et al.*, “Bohatei: Flexible and Elastic DDoS Defense”, in *Proc. USENIX Security*, Washington, D.C., USA, Aug. 2015.

[12] J. Deng, H. Li, H. Hu, *et al.*, "On the Safety and Efficiency of Virtual Firewall Elasticity Control", in *Proc. NDSS*, Jan. 2017.

[13] R. Sommer *et al.*, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection", in *Proc. IEEE S&P*, Oakland, California, USA, May 2010.

[14] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms", *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[15] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-wide Traffic Anomalies", in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 34, 2004, pp. 219–230.

[16] N. Agrawal and S. Tapaswi, "Defense Mechanisms Against DDoS Attacks in a Cloud Computing Environment: State-of-the-Art and Research Challenges", *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3769–3795, 2019.

[17] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks", in *Proc. ESORICS*, Sep. 2005, pp. 319–335.

[18] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic", 2008.

[19] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding Bots in Network Traffic Without Deep Packet Inspection", in *Proc. ACM CoNEXT*, Dec. 2012.

[20] M. Thomas and A. Mohaisen, "Kindred Domains: Detecting and Clustering Botnet Domains Using DNS Traffic", in *Proc. IW3C2*, Seoul, Korea, Apr. 2014.

[21] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee, "Beheading Hydras: Performing Effective Botnet Takedowns", in *Proc. ACM CCS*, Berlin, Germany, Nov. 2013.

[22] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon", in *Proc. ACM IMC*, Jul. 2006, pp. 41–52.

[23] A. Tundis, W. Mazurczyk, and M. Mühlhäuser, "A Review of Network Vulnerabilities Scanning Tools: Types, Capabilities and Functioning", in *Proc. ACM ARES*, Hamburg, Germany, Aug. 2018.

[24] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks", in *Proc. USENIX Security*, San Diego, CA, USA, Aug. 2014.

[25] A. Orebaugh and B. Pinkard, *Nmap in the Enterprise: Your Guide to Network Scanning*. Elsevier, 2011.

[26] Z. Durumeric *et al.*, "ZMap: Fast Internet-wide Scanning and Its Security Applications", in *Proc. USENIX Security*, Washington, D.C. , USA, Aug. 2013.

[27] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier ZMap: Internet-Wide Scanning at 10 Gbps", in *Proc. USENIX WOOT*, San Diego, CA, Dec. 2014.

[28] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning", in *Proc. ACM CCS*, Denver, Colorado, USA, Oct. 2015.

[29] F. Yarochkin, Y. Huang, Y. Hu, and S. Kuo, "Mining Large Network Reconnaissance Data", in *Proc. IEEE PRDC*, Vancouver, BC, Canada, Dec. 2013.

[30] Q. Hu, M. R. Asghar, and N. Brownlee, "Measuring IPv6 DNS Reconnaissance Attacks and Preventing Them Using DNS Guard", in *Proc. IEEE/IFIP DSN*, Luxembourg City, Luxembourg, Jun. 2018.

[31] A. Sridharan and T. Ye, "Tracking Port Scanners on the IP Backbone", in *Proc. ACM LSAD*, Kyoto, Japan, Aug. 2007.

[32] W. Wang, B. Yang, and Y. V. Chen, "Detecting Subtle Port Scans through Characteristics Based on Interactive Visualization", in *Proc. RIIT*, Atlanta, Georgia, USA, Oct. 2014.

[33] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks", *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 12, pp. 2562–2577, 2015.

[34] P. Richter and A. Berger, "Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope", in *Proc. ACM IMC*, Amsterdam, Netherlands, Oct. 2019.

[35] K. Fukuda, J. Heidemann, and A. Qadeer, "Detecting Malicious Activity With DNS Backscatter Over Time", *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3203–3218, Oct. 2017.

[36] K. Fukuda and J. Heidemann, "Who Knocks at the IPv6 Door? Detecting IPv6 Scanning", in *Proc. ACM IMC*, Boston, MA, USA, Oct. 2018.

[37] H. Heo and S. Shin, "Who is Knocking on the Telnet Port: A Large-Scale Empirical Study of Network Scanning", in *Proc. ASIACCS*, Incheon, Republic of Korea, Jun. 2018.

[38] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning", in *Proc. USENIX Security*, San Diego, CA, USA, Aug. 2014.

[39] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity", *ACM Trans. Comput. Syst.*, vol. 24, no. 2, pp. 115–139, May 2006, ISSN: 0734-2071.

[40] G. Gu, R. Perdisci, J. Zhang, W. Lee, *et al.*, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection", in *Proc. USENIX Security*, vol. 5, Jul. 2008, pp. 139–154.

[41] F. Loi, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman, "Systematically Evaluating Security and Privacy for Consumer IoT Devices", in *Proc. IoT S&P*, Dallas, Texas, USA, Nov. 2017.

[42] A. Welzel, C. Rossow, and H. Bos, "On Measuring the Impact of DDoS Botnets", in *Proc. European Workshop on System Security*, Amsterdam, The Netherlands, Apr. 2014.

[43] M. Antonakakis *et al.*, "Understanding the Mirai Botnet", in *Proc. USENIX Security*, Vancouver, BC, USA, Aug. 2017.

[44] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "A Data-Driven Study of DDoS Attacks and Their Dynamics", *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 03, pp. 648–661, May 2020, ISSN: 1941-0018.

[45] W. Chang, A. Mohaisen, A. Wang, and S. Chen, "Measuring Botnets in the Wild: Some New Trends", in *Proc. ASIACCS*, Singapore, Republic of Singapore, Apr. 2015.

[46] M. Karami and D. McCoy, "Understanding the Emerging Threat of DDoS-As-a-Service", in *Proc. USENIX LEET*, Washington, D.C.: USENIX Association, Aug. 2013.

[47] A. Sivanathan, H. H. Gharakheili, F. Loi, *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics", *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, Aug. 2019.

[48] M. Eslahi, M. V. Naseri, H. Hashim, N. M. Tahir, and E. H. M. Saad, "BYOD: Current State and Security Challenges", in *2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, 2014, pp. 189–192.

[49] R. Ogie, "Bring Your Own Device: An Overview of Risk Assessment", *IEEE Consumer Electronics Magazine*, vol. 5, no. 1, pp. 114–119, 2016.

[50] H. M. J. Almohri, L. T. Watson, D. Yao, and X. Ou, "Security Optimization of Dynamic Networks with Probabilistic Graph Modeling and Linear Programming", *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 4, pp. 474–487, 2016.

[51] Cisco Security, *Cisco Firewall Best Practices*, `https://bit.ly/3jQEWtj`, Accessed: 2020-08-31.

[52] M. T. Arefin, M. R. Uddin, N. A. Evan, and M. R. Alam, “Enterprise Network: Security Enhancement and Policy Management Using Next-Generation Firewall (NGFW)”, in *Computer Networks, Big Data and IoT*, Jun. 2021.

[53] E. L. Ngoupé, S. Stoesel, C. Parisot, *et al.*, “A Data Model for Management of Network Device Configuration Heterogeneity”, in *Proc. IEEE/IFIP IM*, Ottawa, Canada, May 2015, pp. 1230–1233.

[54] P. Wang, J. Luo, W. Li, and Y. Qu, “NCPP: A Network Control Programmale Platform of Trustworthy Controllable Network”, in *Proc. IEEE DCSW*, Jun. 2009, pp. 221–226.

[55] A. Starschenko, N. Tcholtchev, A. Prakash, I. Schieferdecker, and R. Chaparadza, “Auto-Configuration of OSPFv3 Routing in Fixed IPv6 Networks”, in *Proc. ICUMT*, Oct. 2015, pp. 196–205.

[56] A. Voronkov, L. H. Iwaya, L. A. Martucci, and S. Lindskog, “Systematic Literature Review on Usability of Firewall Configuration”, *ACM Comput. Surv.*, vol. 50, no. 6, Dec. 2017.

[57] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot, “Management of Stateful Firewall Misconfiguration”, *Comput. Secur.*, vol. 39, pp. 64–85, Nov. 2013, ISSN: 0167-4048.

[58] H. Ismail, H. S. Hamza, and S. M. Mohamed, “Semantic Enhancement for Network Configuration Management”, in *Proc. IEEE GCIoT*, Alexandria, Egypt, Dec. 2018, pp. 1–5.

[59] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner, “Ontology mapping for the interoperability problem in network management”, *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2058–2068, 2005.

[60] H. Ballani and P. Francis, “CONMan: A Step towards Network Manageability”, in *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 2007, pp. 205–216.

[61] A. Wedgbury and K. Jones, “Automated Asset Discovery in Industrial Control Systems: Exploring the Problem”, in *Proc. ICS-CSR*, Ingolstadt, Germany, Sep. 2015, pp. 73–83.

[62] F. Zhu, M. W. Mutka, and L. M. Ni, “Service Discovery in Pervasive Computing Environments”, *IEEE Pervasive Computing*, vol. 4, no. 4, pp. 81–90, Oct. 2005, ISSN: 1536-1268.

[63] T. Bakhshi and B. Ghita, “User Traffic Profiling”, in *Proc. IEEE ITA*, Sep. 2015, pp. 91–97.

[64] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “BLINC: Multilevel Traffic Classification in the Dark”, in *Proc. ACM SIGCOMM*, Philadelphia, Pennsylvania, USA, Oct. 2005.

[65] E. Glatz, “Visualizing Host Traffic through Graphs”, in *Proc. International Symposium on Visualization for Cyber Security*, Ottawa, Ontario, Canada, Sep. 2010, pp. 58–63.

[66] J. J. Pfeiffer III *et al.*, “Attributed Graph Models: Modeling Network Structure with Correlated Attributes”, in *Proc. ACM WWW*, Seoul, Korea, Apr. 2014.

[67] P. Bhattacharya and S. K. Ghosh, “Analytical Framework for Measuring Network Security using Exploit Dependency Graph”, *IET Information Security*, vol. 6, no. 4, pp. 264–270, 2012.

[68] J. Jusko and M. Rehak, “Revealing Cooperating Hosts by Connection Graph Analysis”, in *Proc. SecureComm*, Berlin, Heidelberg: Springer Berlin Heidelberg, Mar. 2013, pp. 241–255.

[69] K. Xu, F. Wang, and L. Gu, “Behavior Analysis of Internet Traffic via Bipartite Graphs and One-Mode Projections”, *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 931–942, Jun. 2014, ISSN: 1063-6692.

[70] A. Jakalan *et al.*, “Social Relationship Discovery of IP Addresses in the Managed IP Networks by Observing Traffic at Network Boundary”, *Comput. Netw.*, vol. 100, no. C, pp. 12–27, May 2016, ISSN: 1389-1286.

[71] N. Poolsappasit, R. Dewri, and I. Ray, “Dynamic Security Risk Management Using Bayesian Attack Graphs”, *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[72] L. Wang *et al.*, “An Attack Graph-Based Probabilistic Security Metric”, in *Data and Applications Security XXII*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[73] M. Albanese, S. Jajodia, and S. Noel, “Time-Efficient and Cost-Effective Network Hardening using Attack Graphs”, in *Proc. IEEE/IFIP DSN*, Jun. 2012, pp. 1–12.

[74] L. Muñoz-González, D. Sgandurra, A. Paudice, and E. C. Lupu, “Efficient Attack Graph Analysis through Approximate Inference”, *ACM Trans. Priv. Secur.*, vol. 20, no. 3, Jul. 2017.

[75] D. Eswaran, C. Faloutsos, S. Guha, and N. Mishra, “SpotLight: Detecting Anomalies in Streaming Graphs”, in *Proc. ACM KDD*, London, United Kingdom, Aug. 2018.

[76] P. Kalmbach, D. Hock, F. Lipp, W. Kellerer, and A. Blenk, “NOracle: Who is Communicating with Whom in My Network?”, in *Proc. ACM SIGCOMM Posters and Demos*, Beijing, China, Aug. 2019.

[77] H. Irshad, G. Ciocarlie, A. Gehani, *et al.*, “TRACE: Enterprise-Wide Provenance Tracking for Real-Time APT Detection”, *IEEE Transactions on Information Forensics and Security*, Sep. 2021.

[78] S. T. Zargar *et al.*, “A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks”, *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, Mar. 2013.

[79] J. Mirkovic, G. Prier, and P. Reiher, “Source-End DDoS Defense”, in *Proc. IEEE NCA*, Apr. 2003, pp. 171–178.

[80] J. Mirkovic and P. Reiher, “D-WARD: A Source-End Defense Against Flooding Denial-of-Service Attacks”, *IEEE Trans. Dependable Secur. Comput.*, vol. 2, no. 3, pp. 216–232, Jul. 2005, ISSN: 1545-5971.

[81] G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson, “A Framework for a Collaborative DDoS Defense”, in *Proc. ACSAC*, Dec. 2006, pp. 33–42.

[82] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, “DDoS Defense by Offense”, *ACM Trans. Comput. Syst.*, vol. 28, no. 1, Aug. 2010.

[83] K. Borders, J. Springer, and M. Burnside, “Chimera: A Declarative Language for Streaming Network Traffic Analysis”, in *Proc. USENIX Security*, Bellevue, WA, Feb. 2012.

[84] S. Narayana *et al.*, “Language-Directed Hardware Design for Network Performance Monitoring”, in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, Aug. 2017.

[85] P. Gao, X. Xiao, D. Li, *et al.*, “SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection”, in *Proc. USENIX Security*, Baltimore, MD, USA, Jul. 2018.

[86] A. Gupta, R. Harrison, M. Canini, *et al.*, “Sonata: Query-driven Streaming Network Telemetry”, in *Proc. ACM SIGCOMM*, Budapest, Hungary, Aug. 2018.

[87] M. R. Lyu and L. K. Y. Lau, “Firewall Security: Policies, Testing and Performance Evaluation”, in *Proc. COMPSAC*, Oct. 2000, pp. 116–121.

[88] A. Wool, “A Quantitative Study of Firewall Configuration Errors”, *Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.

[89] K. Salah, K. Elbadawi, and R. Boutaba, “Performance Modeling and Analysis of Network Firewalls”, *IEEE Transactions on Network and Service Management*, vol. 9, no. 1, pp. 12–21, Mar. 2012.

[90] C. Wang, D. Zhang, H. Lu, *et al.*, “An Experimental Study on Firewall Performance: Dive into the Bottleneck for Firewall Effectiveness”, in *Proc. International Conference on Information Assurance and Security*, Nov. 2014, pp. 71–76.

[91] M. Aimon *et al.*, “Performance Evaluations of IPTables Firewall Solutions under DDoS attacks”, vol. 11, Dec. 2015.

[92] U. Mustafa, M. M. Masud, Z. Trabelsi, T. Wood, and Z. A. Harthi, "Firewall Performance Optimization using Data Mining Techniques", in *Proc. IWCMC*, Jul. 2013.

[93] A. K. Vasu *et al.*, "Improving Firewall Performance by Eliminating Redundancies In Access Control Lists", *International Journal of Computer Networks*, vol. 6, pp. 92–107, 5 Sep. 2014.

[94] H. Tegenaw and M. Kifle, "Application Aware Firewall Architecture to Enhance Performance of Enterprise Network", in *Proc. IEEE AFRICON*, Sep. 2015, pp. 1–10.

[95] P. J. Lee, H. Guo, and B. Veeravalli, "Enhancing CII firewall performance through hash based rule lookup", in *Proc. IEEE TENCON*, Nov. 2017, pp. 2285–2290.

[96] ESNet, *Firewall Performance Issues*, `https://fasterdata.es.net/network-tuning/firewall-performance-issues/`, Accessed: 2017-09-01, 2018.

[97] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", *Comput. Netw.*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999, ISSN: 1389–1286.

[98] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks.", in *Lisa*, vol. 99, 1999, pp. 229–238.

[99] C. Kreibich and J. Crowcroft, "Honeycomb: Creating Intrusion Detection Signatures Using Honeypots", *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, Jan. 2004.

[100] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms", in *Proc. IEEE S&P*, May 2005, pp. 226–241.

[101] Y. Xie, F. Yu, K. Achan, *et al.*, "Spamming Botnets: Signatures and Characteristics", *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 171–182, Aug. 2008.

[102] S. Patton, W. Yurcik, and D. Doss, "An Achilles? Heel in Signature-based IDS: Squealing False Positives in SNORT", in *Proc. RAID*, Oct. 2001.

[103] R. Sommer and V. Paxson, "Enhancing Byte-Level Network Intrusion Detection Signatures with Context", in *Proc. ACM CCS*, Washington D.C., USA, 2003, pp. 262–271.

[104] D. Day and B. Burns, "A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines", in *Proc. ICDS*, Feb. 2011, pp. 187–192.

[105] E. Albin and N. C. Rowe, "A Realistic Experimental Comparison of the Suricata and Snort intrusion-detection systems", in *Proc. WAINA*, Mar. 2012, pp. 122–127.

[106] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, “Network Intrusion Detection for IoT Security Based on Learning Techniques”, *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019.

[107] E. Benkhelifa, T. Welsh, and W. Hamouda, “A Critical Review of Practices and Challenges in Intrusion Detection Systems for IoT: Toward Universal and Resilient Systems”, *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3496–3509, 2018.

[108] G. Liu, K. K. Ramakrishnan, M. Schlansker, J. Tourrilhes, and T. Wood, “Design Challenges for High Performance, Scalable NFV Interconnects”, in *Proc. the Workshop on Kernel-Bypass Networks*, Los Angeles, CA, USA, Aug. 2017.

[109] M. Vallentin, R. Sommer, J. Lee, *et al.*, “The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware”, in *Proc. RAID*, Gold Goast, Australia, Sep. 2007, pp. 107–126.

[110] L. De Carli, R. Sommer, and S. Jha, “Beyond Pattern Matching: A Concurrency Model for Stateful Deep Packet Inspection”, in *Proc. ACM CCS*, Scottsdale, Arizona, USA, Nov. 2014, pp. 1378–1390.

[111] S. Kong, R. Smith, and C. Estan, “Efficient Signature Matching with Multiple Alphabet Compression Tables”, in *Proc. SecureComm*, Istanbul, Turkey, Sep. 2008.

[112] X. Yu, W. Feng, D. Yao, and M. Becchi, “O3FA: A Scalable fFinite Automata-Based Pattern-Matching Engine for Out-of-Order Deep Packet Inspection”, in *Proc. ACM/IEEE ANCS*, Mar. 2016, pp. 1–11.

[113] H. Li, H. Hu, G. Gu, G.-J. Ahn, and F. Zhang, “vNIDS: Towards Elastic Security with Safe and Efficient Virtualization of Network Intrusion Detection Systems”, in *Proc. ACM CCS*, Toronto, Canada, Oct. 2018.

[114] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, “Kronecker Graphs: An Approach to Modeling Networks”, *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, Mar. 2010, ISSN: 1532-4435.

[115] J. J. Villalobos, I. Rodero, and M. Parashar, “An Unsupervised Approach for Online Detection and Mitigation of High-Rate DDoS Attacks Based on an In-Memory Distributed Graph Using Streaming Data and Analytics”, in *Proc. IEEE/ACM BDCAT*, Austin, Texas, USA, Dec. 2017.

[116] M. Callau-Zori, R. Jiménez-Peris, V. Gulisano, *et al.*, “STONE: A Stream-Based DDoS Defense Framework”, in *Proc. ACM SAC*, Coimbra, Portugal, Mar. 2013.

[117] A. McGregor, “Graph Stream Algorithms: A Survey”, *SIGMOD Rec.*, vol. 43, no. 1, pp. 9–20, May 2014.

[118] D. Ippoliti, C. Jiang, Z. Ding, and X. Zhou, “Online Adaptive Anomaly Detection for Augmented Network Flows”, *ACM Trans. Auton. Adapt. Syst.*, vol. 11, no. 3, Sep. 2016.

[119] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, “A Novel PCA-Based Network Anomaly Detection”, in *Proc. IEEE ICC*, Jun. 2011, pp. 1–5.

[120] F. Iglesias and T. Zseby, “Analysis of Network Traffic Features for Anomaly Detection”, *Machine Learning*, vol. 101, no. 1, pp. 59–84, 2015.

[121] C. Ma, X. Du, and L. Cao, “Analysis of Multi-Types of Flow Features Based on Hybrid Neural Network for Improving Network Anomaly Detection”, *IEEE Access*, vol. 7, pp. 148 363–148 380, 2019.

[122] S. Jin *et al.*, “A Covariance Analysis Model for DDoS Attack Detection”, in *Proc. IEEE ICC*, Paris, France, Jun. 2004.

[123] K. Lee, J. Kim, K. Kwon, Y. Han, and S. Kim, “DDoS Attack Detection Method Using Cluster Analysis”, *Expert Systems with Applications*, vol. 34, pp. 1659–1665, Apr. 2008.

[124] H. Rahmani, N. Sahli, and F. Kamoun, “DDoS Flooding Attack Detection Scheme Based on F-Divergence”, *Computer Communications*, vol. 35, no. 11, pp. 1380–1391, 2012.

[125] L. Nobach, O. Hohlfeld, and D. Hausheer, “New Kid on the Block: Network Functions Visualization: From Big Boxes to Carrier Clouds”, *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 3, 7:1–7:8, Jul. 2018, ISSN: 0146-4833.

[126] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN: An Intellectual History of Programmable Networks”, 2013.

[127] S. Sezer, S. Scott-Hayward, P. K. Chouhan, *et al.*, “Are We Ready for SDN? Implementation Challenges for Software-Defined Networks”, *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[128] R. Sommer, V. Paxson, and N. Weaver, “An Architecture for Exploiting Multi-Core Processors to Parallelize Network Intrusion Prevention”, *Concurr. Comput. : Pract. Exper.*, vol. 21, no. 10, pp. 1255–1279, Jul. 2009.

[129] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella, “Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr”, in *Proc. USENIX NSDI*, Santa Clara, CA, Mar. 2016, pp. 239–253.

[130] A. Panda, S. Han, K. Jang, *et al.*, “NetBricks: Taking the V out of NFV”, in *Proc. USENIX OSDI*, Savannah, GA, USA, Nov. 2016.

[131] W. Zhang, G. Liu, W. Zhang, *et al.*, “OpenNetVM: A Platform for High Performance Network Service Chains”, in *Proc. ACM HotMiddlebox*, Florianopolis, Brazil, Aug. 2016, pp. 26–31.

[132] Y. Zhang, B. Anwer, V. Gopalakrishnan, *et al.*, “ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining”, in *Proc. ACM SOSR*, Santa Clara, CA, USA, Apr. 2017, pp. 143–149.

[133] M. Kablan, A. Alsudais, E. Keller, and F. Le, “Stateless Network Functions: Breaking the Tight Coupling of State and Processing”, in *Proc. USENIX NSDI*, Boston, MA, USA, Mar. 2017, pp. 97–112.

[134] R. Braga, E. Mota, and A. Passito, “Lightweight DDoS Flooding Attack Detection using NOX/OpenFlow”, in *Proc. IEEE LCN*, Oct. 2010, pp. 408–415.

[135] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, “A SDN-Oriented DDoS Blocking Scheme for Botnet-Based Attacks”, in *Proc. IEEE ICUFN*, Jul. 2014, pp. 63–68.

[136] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments”, *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014, ISSN: 1389-1286.

[137] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, “Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags”, in *Proc. USENIX NSDI*, Seattle, WA, Apr. 2014, pp. 533–546.

[138] C. Yoon, T. Park, S. Lee, *et al.*, “Enabling Security Functions with SDN”, *Comput. Netw.*, vol. 85, no. C, pp. 19–35, Jul. 2015, ISSN: 1389-1286.

[139] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, “Atlantic: A framework for anomaly traffic detection, classification, and mitigation in SDN”, in *Proc. IEEE/IFIP NOMS*, Apr. 2016, pp. 27–35.

[140] C. Liu, A. Raghuramu, C.-N. Chuah, and B. Krishnamurthy, “Piggybacking Network Functions on SDN Reactive Routing: A Feasibility Study”, in *Proc. ACM SOSR*, Santa Clara, CA, USA, Apr. 2017.

[141] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges”, *Computers & Security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[142] A. L. Buczak and E. Guven, “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection”, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[143] W. Lee and S. J. Stolfo, “A Framework for Constructing Features and Models for Intrusion Detection Systems”, *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 227–261, Nov. 2000, ISSN: 1094-9224.

[144] J. Shun and H. A. Malki, “Network intrusion detection system using neural networks”, in *Proc. ICNC*, vol. 5, Oct. 2008, pp. 242–246.

[145] L. Koc, T. A. Mazzuchi, and S. Sarkani, “A Network Intrusion Detection System Based on A Hidden Na´ıve Bayes Multiclass Classifier”, *Expert Systems with Applications*, vol. 39, no. 18, pp. 13 492–13 500, 2012.

[146] M. Javed and V. Paxson, “Detecting Stealthy, Distributed SSH Bruteforcing”, in *Proc. ACM CCS*, Berlin, Germany, Nov. 2013.

[147] A. K. Shrivas and A. K. Dewangan, “An Ensemble Model for Classification of Attacks with Feature Selection Based on KDD99 and NSL-KDD Data Set”, *International Journal of Computer Applications*, vol. 99, no. 15, 2014.

[148] C. Hsieh and T. Chan, “Detection DDoS Attacks Based on Neural-Network using Apache Spark”, in *Proc. ICASI*, May 2016.

[149] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning”, in *Proc. ACM CCS*, Dallas, Texas, USA, Nov. 2017.

[150] H. Siadati and N. Memon, “Detecting Structurally Anomalous Logins Within Enterprise Networks”, in *Proc. ACM CCS*, Dallas, Texas, USA, Nov. 2017.

[151] D. Tang, J. Chen, X. Wang, S. Zhang, and Y. Yan, “A New Detection Method for LDoS Attacks based on Data Mining”, *Future Generation Computer Systems*, vol. 128, pp. 73–87, Mar. 2022.

[152] J. Jang-Jaccard and S. Nepal, “A Survey of Emerging Threats in Cybersecurity”, *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 973–993, 2014.

[153] M. Umer, M. S. Ramzan, and Y. Bi, “Flow-Based Intrusion Detection: Techniques and Challenges”, *Computers & Security*, vol. 70, Jun. 2017.

[154] K. Neupane, R. Haddad, and L. Chen, “Next Generation Firewall for Network Security: A Survey”, in *Proc. IEEE SoutheastCon*, Tampa Bay Area, FL, USA, Apr. 2018, pp. 1–6.

[155] C. Chen *et al.*, “A Survey of Network Security Situational Awareness Technology”, in *Proc. ICAIS*, New York, NY, USA, Jul. 2019.

[156] S. Sengupta, A. Chowdhary, A. Sabur, *et al.*, “A Survey of Moving Target Defenses for Network Security”, *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.

[157] X. Chen, K. Makki, K. Yen, and N. Pissinou, “Sensor Network Security: A Survey”, *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 52–73, 2009.

[158] S. Scott-Hayward, S. Natarajan, and S. Sezer, “A Survey of Security in Software Defined Networks”, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[159] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in Software Defined Networks: A Survey", *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[160] Y. Al-Hadhrami and F. Hussain, "DDoS Attacks in IoT Networks: a Comprehensive Systematic Literature Review", *World Wide Web*, May 2021.

[161] J. Cao, M. Ma, H. Li, *et al.*, "A Survey on Security Aspects for 3GPP 5G Networks", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 170–195, 2020.

[162] A. Bhardwaj, V. Mangat, R. Vig, S. Halder, and M. Conti, "Distributed Denial of Service Attacks in Cloud: State-of-the-Art of Scientific and Commercial Solutions", *Computer Science Review*, Feb. 2021.

[163] S. Marshall, "CANDID: Classifying Assets in Networks by Determining Importance and Dependencies", University of California at Berkeley, Electrical Engineering and Computer Sciences, Tech. Rep., May 2013.

[164] Deloitte. (2018). Elevating Cybersecurity on the Higher Education Leadership Agenda.

[165] EfficientIP, "A New Era Of Network Attacks", Global DNS Threat Report, 2018.

[166] J. Vijayan, *Frequency & Costs of DNS-Based Attacks Soar*, `https://ubm.io/2Nxx5Cr`, Accessed: 2018-05-16, 2018.

[167] C. Fachkha, E. Bou-Harb, and M. Debbabi, "Fingerprinting Internet DNS Amplification DDoS Activities", in *Proc. IEEE/IFIP NOMS*, Mar. 2014.

[168] CISA, *Alert (TA13-088A) DNS Amplification Attacks*, `https://www.us-cert.gov/ncas/alerts/TA13-088A`, Accessed: 2018-05-01, 2018.

[169] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS Amplification Attack Revisited", *Comput. Secur.*, vol. 39, pp. 475–485, Nov. 2013, ISSN: 0167-4048.

[170] Cisco Blog, *Overcoming the DNS Blind Spot*, `https://blogs.cisco.com/security/overcoming-the-dns-blind-spot`, Accessed: 2019-05-15, 2016.

[171] X. Ma, J. Zhang, J. Tao, *et al.*, "DNSRadar: Outsourcing Malicious Domain Detection Based on Distributed Cache-Footprints", *IEEE TIFS*, vol. 9, no. 11, pp. 1906–1921, Nov. 2014.

[172] M. Antonakakis, R. Perdisci, Y. Nadji, *et al.*, "From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware", in *Proc. USENIX Security*, Bellevue, WA, USA, Aug. 2012.

[173] H. Gao et al., "Reexamining DNS From a Global Recursive Resolver Perspective", *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 43–57, Feb. 2016.

[174] Y. Chen *et al.*, “DNS Noise: Measuring the Pervasiveness of Disposable Domains in Modern DNS Traffic”, in *Proc. IEEE/IFIP DSN*, Atlanta, Georgia, USA, Jun. 2014.

[175] S. Hao, A. Kantchelian, B. Miller, V. Paxson, and N. Feamster, “PREDATOR: Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration”, in *Proc. ACM CCS*, Oct. 2016.

[176] Y. Lee and N. Spring, “Identifying and Analyzing Broadband Internet Reverse DNS Names”, in *Proc. ACM CoNEXT*, Incheon, Republic of Korea, Dec. 2017.

[177] J. Ahmed, H. Habibi Gharakheili, C. Russell, and V. Sivaraman, “Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks”, in *Proc. IFIP/IEEE IM*, Washington DC, USA, Apr. 2019.

[178] M. Almeida, A. Finamore, D. Perino, N. Vallina-Rodriguez, and M. Varvello, “Dissecting DNS Stakeholders in Mobile Networks”, in *Proc. ACM CoNEXT*, Incheon, Republic of Korea, Dec. 2017.

[179] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, “FANCI : Feature-based Automated NXDomain Classification and Intelligence”, in *Proc. USENIX Security*, Baltimore, MD, USA, Aug. 2018.

[180] M. Müller, G. C. M. Moura, R. de O. Schmidt, and J. Heidemann, “Recursives in the Wild: Engineering Authoritative DNS Servers”, in *Proc. ACM IMC*, London, United Kingdom, Nov. 2017.

[181] S. Hao, N. Feamster, and R. Pandrangi, “Monitoring the Initial DNS Behavior of Malicious Domains”, in *Proc. ACM IMC*, Berlin, Germany, Nov. 2011.

[182] T. Chung *et al.*, “Understanding the Role of Registrars in DNSSEC Deployment”, in *Proc. ACM IMC*, London, UK, Nov. 2017.

[183] D. C. MacFarland, C. A. Shue, and A. J. Kalafut, “Characterizing Optimal DNS Amplification Attacks and Effective Mitigation”, in *Proc. Springer PAM*, New Tork, NY, USA, Mar. 2015.

[184] D. C. MacFarland *et al.*, “The Best Bang for the Byte: Characterizing the Potential of DNS Amplification Attacks”, *Computer Networks*, vol. 116, pp. 12–21, Apr. 2017.

[185] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, “DNSSEC and Its Potential for DDoS Attacks: A Comprehensive Measurement Study”, in *Proc. ACM IMC*, Vancouver, BC, Canada, Nov. 2014.

[186] H. Choi and H. Lee., “Identifying Botnets by Capturing Group Activities in DNS Traffic”, *Computer Networks*, vol. 56, no. 1, pp. 20–33, Feb. 2012.

[187] D. Yang, Z. Li, and G. Tyson, “A Deep Dive into DNS Query Failures”, in *Proc. USENIX ATC*, Virtual Event, Jul. 2020.

[188] W. Rweyemamu et al., "Clustering and the Weekend Effect: Recommendations for the Use of Top Domain Lists in Security Research", in *Proc. Springer PAM*, Puerto Varas, Chile, Mar. 2019.

[189] Y. Fu et al., "Stealthy Domain Generation Algorithms", *IEEE TIFS*, vol. 12, no. 6, pp. 1430–1443, Jun. 2017.

[190] IETF, *DNS Security Introduction and Requirements*, `https://www.ietf.org/rfc/rfc4033.txt`, Accessed: 2018-05-28, 2018.

[191] S. Son and V. Shmatikov, "The Hitchhiker's Guide to DNS Cache Poisoning", in *Proc. SecureComm*, Sep. 2010.

[192] P. Hoffman and P. McManus, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310. [Online]. Available: `https://tools.ietf.org/html/rfc8310`.

[193] P. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484. [Online]. Available: `https://tools.ietf.org/html/rfc8484`.

[194] DPDK Project, *Developer Quick Start Guide Learn How To Get Involved With DPDK*, `https://www.dpdk.org`, Accessed: 2020-01-24, 2020.

[195] M. Bykova, S. Ostermann, and B. Tjaden, "Detecting Network Intrusions via a Statistical Analysis of Network Packet Characteristics", in *Proc. Southeastern Symposium on System Theory*, Mar. 2001.

[196] M. Bykova and S. Ostermann, "Statistical Analysis of Malformed Packets and Their Origins in the Modern Internet", in *Proc. ACM IMC*, Marseille, France, Jul. 2002.

[197] H. Gao, V. Yegneswaran, Y. Chen, *et al.*, "An Empirical Reexamination of Global DNS Behavior", in *Proc. ACM SIGCOMM*, Hong Kong, China, Aug. 2013.

[198] M. Anagnostopoulos, G. Kambourakis, S. Gritzalis, and D. K. Y. Yau, "Never Say Never: Authoritative TLD Nameserver-Powered DNS Amplification", in *Proc. IEEE/IFIP NOMS*, Apr. 2018.

[199] Cloudflare, *What Happened Next: The Deprecation of ANY*, `https://blog.cloudflare.com/what-happened-next-the-deprecation-of-any/`, Accessed: 2019-6-17, 2019.

[200] J. Abley *et al.*, "Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY", RFC 8482. [Online]. Available: `https://tools.ietf.org/html/rfc8482`.

[201] J. Ahmed, H. Habibi Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts", *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 265–279, 2020.

[202] Palo Alto Networks, *DoS and Zone Protection Best Practices*, `https://bit.ly/2HQOMwU`, Accessed: 2018-28-1, 2018.

[203] Fortinet, *FortiDDoS and Verisign DDoS Protection Service*, `https://bit.ly/2DsDObH`, Accessed: 2018-11-2, 2018.

[204] Cisco Systems, *Protection Against Distributed Denial of Service Attacks*, `https://bit.ly/2WUbvvK`, Accessed: 2018-11-2, 2018.

[205] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A Coding Approach to Event Correlation", in *Proc. International Symposium on Integrated Network Management*, Santa Barbara, CA, USA, May 1995.

[206] T. Benson, A. Akella, and D. Maltz, "Unraveling the Complexity of Network Management", in *Proc. NSDI*, Boston, Massachusetts, USA, Apr. 2009.

[207] M. Lyu, D. Sherratt, A. Sivanathan, *et al.*, "Quantifying the Reflective DDoS Attack Capability of Household IoT Devices", in *Proc. ACM WiSec*, Boston, MA, USA, Jul. 2017.

[208] C. Brignone. et al., "Real Time Asset Tracking in the Data Center", *Distributed Parallel Databases*, vol. 21, pp. 145–165, Jun. 2007.

[209] M. Mesarina. et al., "Automating Server Tracking for Data Centers", in *Proc. ACM SenSys*, Baltimore, Maryland, Nov. 2004.

[210] Balbix, *The 10 Most Common Cybersecurity Posture Blindspots*, `https://bit.ly/3s9CNNO`, Accessed: 2020-11-26, 2020.

[211] K. Jackson, *What Causes Network Blind Spots—and How to Prevent Them*, `https://www.helpsystems.com/blog/what-causes-network-blind-spots-and-how-prevent-them`, Accessed: 2020-11-26, 2019.

[212] B. Caswell, J. C. Foster, R. Russell, J. Beale, and J. Posluns, *Snort 2.0 Intrusion Detection.* Syngress Publishing, 2003.

[213] M. Casado, T. Garfinkel, A. Akella, *et al.*, "SANE: A Protection Architecture for Enterprise Networks", in *Proc. USENIX Security*, Vancouver, B.C., Canada, Jul. 2006.

[214] A. Zand, A. Houmansadr, G. Vigna, R. Kemmerer, and C. Kruegel, "Know Your Achilles' Heel: Automatic Detection of Network Critical Services", in *Proc. ACSAC*, Los Angeles, CA, USA, Dec. 2015.

[215] G. Tan, M. Poletto, J. Guttag, and F. Kaashoek, "Role Classification of Hosts within Enterprise Networks Based on Connection Patterns", in *Proc. USENIX ATEC*, San Antonio, Texas, Jun. 2003.

[216] T.-F. Yen, A. Oprea, K. Onarlioglu, *et al.*, "Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks", in *Proc. ACSAC*, New Orleans, Louisiana, USA, Dec. 2013.

[217] S. E. Yusuf, M. Ge, J. B. Hong, *et al.*, “Security Modelling and Analysis of Dynamic Enterprise Networks”, in *Proc. IEEE CIT*, Nadi, Fiji, Dec. 2016.

[218] A. Dainotti, A. Pescape, and K. C. Claffy, “Issues and Future Directions in Traffic Classification”, *IEEE Network*, vol. 26, no. 1, pp. 35–40, Jan. 2012.

[219] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos, “Is P2P Dying or Just Hiding?”, in *Proc. IEEE GLOBECOM*, Dallas, TX, USA, Nov. 2004.

[220] S. Alcock, J.-P. Möller, and R. Nelson, “Sneaking Past the Firewall: Quantifying the Unexpected Traffic on Major TCP and UDP Ports”, in *Proc. ACM IMC*, Santa Monica, California, USA, Nov. 2016.

[221] R. Boutaba, M. A. Salahuddin, N. Limam, *et al.*, “A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities”, *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun. 2018.

[222] E. Papadogiannaki, C. Halevidis, P. Akritidis, and L. Koromilas, “OTTer: A Scalable High-Resolution Encrypted Traffic Identification Engine”, in *Proc. RAID*, Heraklion, Crete, Greece, Sep. 2018.

[223] D. Apiletti, E. Baralis, T. Cerquitelli, *et al.*, “SeLINA: A Self-Learning Insightful Network Analyzer”, *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 696–710, Sep. 2016, ISSN: 1932-4537.

[224] M. H. Mazhar and Z. Shafiq, “Real-time Video Quality of Experience Monitoring for HTTPS and QUIC”, in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, Apr. 2018.

[225] H. Zhang, L. Chen, B. Yi, *et al.*, “CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark”, in *Proc. ACM SIGCOMM*, Florianopolis, Brazil, Aug. 2016.

[226] B. Li, M. H. Gunes, G. Bebis, and J. Springer, “A Supervised Machine Learning Approach to Classify Host Roles on Line Using sFlow”, in *Proc. ACM HPPN*, New York, USA, Jun. 2013.

[227] J. Kohout and T. Pevný, “Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test”, *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 788–801, Mar. 2018, ISSN: 1556-6013.

[228] X. Hu, J. Jang, M. P. Stoecklin, *et al.*, “BAYWATCH: Robust Beaconing Detection to Identify Infected Hosts in Large-Scale Enterprise Networks”, in *Proc. IEEE/IFIP DSN*, Toulouse, France, Jun. 2016.

[229] T. Yang, J. Jiang, P. Liu, *et al.*, “Elastic Sketch: Adaptive and Fast Network-wide Measurements”, in *Proc. ACM SIGCOMM*, Budapest, Hungary, Aug. 2018.

[230] Z. Liu, R. Ben-Basat, G. Einziger, *et al.*, “Nitrosketch: Robust and General Sketch-Based Monitoring in Software Switches”, in *Proc. ACM SIGCOMM*, Beijing, China, Aug. 2019.

[231] H. Habibi Gharakheili, M. Lyu, Y. Wang, H. Kumar, and V. Sivaraman, “iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification”, *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1071–1085, Sep. 2019.

[232] T. Yu *et al.*, “PSI: Precise Security Instrumentation for Enterprise Networks”, in *Proc. NDSS*, San Diego, CA, USA, Feb. 2017.

[233] J. Touch *et al.*, *Service Name and Transport Protocol Port Number Registry*, `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`, Accessed: 2019-03-24, 2019.

[234] I. Livadariu, K. Benson, A. Elmokashfi, A. Dhamdhere, and A. Dainotti, “Inferring Carrier-Grade NAT Deployment in the Wild”, in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, Oct. 2018.

[235] PCI Security Standards Council LLC, “The Prioritized Approach to Pursue PCI DSS Compliance”, Tech. Rep., 2018.

[236] Akamai Technologies, *How Securing Recursive DNS Proactively Protects Your Network*, `https://blogs.akamai.com/2017/10/how-securing-recursive-dns-proactively-protects-your-network.html`, Accessed: 2021-05-14, 2017.

[237] M. Keshk, E. Sitnikova, N. Moustafa, J. Hu, and I. Khalil, “An Integrated Framework for Privacy-Preserving Based Anomaly Detection for Cyber-Physical Systems”, *IEEE Transactions on Sustainable Computing*, vol. 6, no. 1, pp. 66–79, 2021.

[238] D. Berrar and W. Dubitzky, “Information gain”, in *Encyclopedia of Systems Biology*, W. Dubitzky, O. Wolkenhauer, K.-H. Cho, and H. Yokota, Eds. New York, NY: Springer New York, 2013, pp. 1022–1023.

[239] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, “Microservices: The Journey So Far and Challenges Ahead”, *IEEE Software*, vol. 35, no. 03, pp. 24–35, May 2018, ISSN: 0740-7459.

[240] NoviFlow, *NoviSwitch™ 2122 High Performance OpenFlow Switch*, `https://noviflow.com/wp-content/uploads/NoviSwitch-2122-Datasheet-1.pdf`, Accessed: 2018-28-1, 2018.

[241] L. Q. Nguyen, *Build Performant Network Functions with NFF-Go*, `https://intel.ly/2PcR3XB`, Accessed: 2019-06-28, 2019.

[242] Zeek, *The Zeek Network Security Monitor*, `https://www.zeek.org`, Accessed: 2019-11-21, 2019.

[243] Nmap, *TCP ACK Scan*, `https://nmap.org/book/scan-methods-ack-scan.html`, Accessed: 2019-08-29, 2019.

[244] Red Button DDoS Experts, *ACK Flood*, `https://www.red-button.net/ddos-glossary/ack-flood/`, Accessed: 2019-05-10, 2019.

[245] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse", in *Proc. NDSS*, San Diego, CA, USA, Feb. 2014.

[246] Akamai Technologies, *Threat Advisory: Mirai Botnet*, `https://bit.ly/2UC1JfJ`, Accessed: 2017-12-3, 2017.

[247] Y. M. P. Pa *et al.*, "IoTPOT: Analysing the Rise of IoT Compromises", in *Proc. USENIX WOOT*, Washington, D.C., USA, Aug. 2015.

[248] T. Greene, *How the Dyn DDoS attack unfolded*, `https://www.networkworld.com/article/3134057/how-the-dyn-ddos-attack-unfolded.html`, Accessed: 2020-11-03, 2016.

[249] EfficientIP, "EfficientIP and IDC: DNS Attacks Cost Nearly $1 Million Each, Increasingly Impacting the Cloud", Global DNS Threat Report, 2020.

[250] D. Dagon *et al.*, "Recursive DNS Architectures and Vulnerability Implications", in *Proc. NDSS*, San Diego, CA, USA, Feb. 2009.

[251] J. Bushart, "Optimizing Recurrent Pulsing Attacks using Application-Layer Amplification of Open DNS Resolvers", in *Proc. USENIX WOOT*, Baltimore, MD, USA, Aug. 2018.

[252] J. Krupp *et al.*, "Identifying the Scan and Attack Infrastructures Behind Amplification DDoS Attacks", in *Proc. ACM CCS*, Vienna, Austria, Oct. 2016.

[253] Sophos Group, *Sophos XG Firewall: How to prevent DoS and DDoS attacks*, `https://bit.ly/2tcOPZY`, Accessed: 2018-11-2, 2018.

[254] W. Meng *et al.*, "Rampart: Protecting Web Applications from CPU-exhaustion Denial-of-service Attacks", in *Proc. USENIX Security*, Baltimore, MD, USA, Aug. 2018.

[255] Symantec, *The continued rise of DDoS attacks*, `https://bit.ly/2UFeWqK`, Accessed: 2019-4-2, 2019.

[256] A. Wang, A. Mohaisen, W. Chang, and S. Chen, "Delving into Internet DDoS Attacks by Botnets: Characterization and Analysis", in *Proc. IEEE/IFIP DSN*, Rio de Janeiro, Brazil, Jun. 2015.

[257] B. Liu *et al.*, "Who Is Answering My Queries: Understanding and Characterizing Interception of the DNS Resolution Path", in *Proc. USENIX Security*, Baltimore, MD, USA, Aug. 2018.

[258] D. Liu, S. Hao, and H. Wang, "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records", in *Proc. ACM CCS*, Vienna, Austria, Oct. 2016.

[259] M. Kührer *et al.*, "Going Wild: Large-Scale Classification of Open DNS Resolvers", in *Proc. ACM IMC*, Tokyo, Japan, Oct. 2015.

[260] B. Zdrnja, N. Brownlee, and D. Wessels, "Passive Monitoring of DNS Anomalies", in *Proc. DIMVA*, Lucerne, Switzerland, Jul. 2007.

[261] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, "Detecting Malware Domains at the Upper DNS Hierarchy", in *Proc. USENIX Security*, Berkeley, CA, USA, Aug. 2011.

[262] R. Arends, R. Austein, D. M. M. Larson, and R.Rose, "DNS Security Introduction and Requirements", RFC 4033, Mar. 2005, pp. 1–56. [Online]. Available: `https://www.ietf.org/rfc/rfc4033.txt`.

[263] K. Bhardwaj *et al.*, "Towards IoT-DDoS Prevention Using Edge Computing", in *Proc. USENIX HotEdge*, Boston, MA, USA, Aug. 2018.

[264] V. Sekar *et al.*, "LADS: Large-scale Automated DDOS Detection System", in *Proc. USENIX ATC*, Boston, MA, USA, May 2006.

[265] M. Zhang, G. Li, L. Xu, *et al.*, "Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures", in *Proc. Springer RAID*, Heraklion, Crete, Greece, Sep. 2018.

[266] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes", in *Proc. IFIP/IEEE IM*, Arlington, VA, USA, Apr. 2019.

[267] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, "MiddlePolice: Toward Enforcing Destination-Defined Policies in the Middle of the Internet", in *Proc. ACM CCS*, Vienna, Austria, Oct. 2016.

[268] K. Du, H. Yang, Z. Li, H. Duan, and K. Zhang, "The Ever-Changing Labyrinth: A Large-Scale Analysis of Wildcard DNS Powered Blackhat SEO", in *Proc. USENIX Security*, Austin, TX, USA, Aug. 2016.

[269] N. Daswani and H. Garcia-Molina, "Query-flood DoS Attacks in Gnutella", in *Proc. ACM CCS*, Washington, DC, USA, Nov. 2002.

[270] J. Loveless, S. Stoikov, and R. Waeber, "Online Algorithms in High-frequency Trading", *ACM Queue*, vol. 11, no. 8, 30:30–30:41, Aug. 2013.

[271] B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products", *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[272] F. T. Liu *et al.*, "Isolation Forest", in *Proc. IEEE ICDM*, Pisa, Italy, Dec. 2008.

[273] L. M. Manevitz and M. Yousef, "One-class Svms for Document Classification", *J. Mach. Learn. Res.*, vol. 2, pp. 139–154, Mar. 2002, ISSN: 1532-4435.

[274] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-Dimensional and Large-Scale Anomaly Detection using a Linear One-Class SVM with Deep Learning", *Pattern Recognition*, vol. 58, pp. 121–134, Apr. 2016.

[275] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-Based Anomaly Detection", *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, Mar. 2012.

[276] Palo AIto Networks, *PA-3000 Series Datasheet*, `https://bit.ly/2MPcsk2`, Accessed: 2018-28-1, 2018.

[277] H. Asghari, *Offline IP address to Autonomous System Number lookup module*, `https://pypi.org/project/pyasn/`, Accessed: 2020-11-05, 2020.

[278] Symantec Corporation, *IP Reputation Investigation*, `https://ipremoval.sms.symantec.com/`, Accessed: 2018-28-1, 2018.

[279] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and Other Botnets", *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[280] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving Into Internet DDoS Attacks by Botnets: Characterization and Analysis", *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, Dec. 2018.

[281] A. Abhishta, R. van Rijswijk-Deij, and L. J. M. Nieuwenhuis, "Measuring the Impact of a Successful DDoS Attack on the Customer Behaviour of Managed DNS Service Providers", *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 5, Jan. 2019.

[282] Akamai Technologies, *2019 State of the Internet Security: DDoS and Application Attacks*, `https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/state-of-the-internet-security-ddos-and-application-attacks-2019.pdf`, Accessed: 2019-10-08, 2019.

[283] Blackhat, *DDoS Protection Bypass Techniques*, `https://media.blackhat.com/us-13/US-13-Nixon-Denying-Service-to-DDOS-Protection-Services-WP.pdf`, Accessed: 2019-10-08, 2019.

[284] Forcepoint, *Attacking the Internal Network from the Public Internet using a Browser as a Proxy*, `https://bit.ly/3ndDtRT`, Accessed: 2019-10-08, 2019.

[285] S. Haas and M. Fischer, "GAC: Graph-Based Alert Correlation for the Detection of Distributed Multi-Step Attacks", in *Proc. ACM SAC*, Pau, France, Apr. 2018.

[286] K. Singh, P. Singh, and K. Kumar, "Application Layer HTTP-GET Flood DDoS Attacks: Research Landscape and Challenges", *Computers & Security*, vol. 65, pp. 344–372, 2017.

[287] M. H. Bhuyan, D. Bhattacharyya, and J. Kalita, "An Empirical Evaluation of Information Metrics for Low-Rate and High-Rate DDoS Attack Detection", *Pattern Recognition Letters*, vol. 51, p. 7, 2015.

[288] K. Hong, Y. Kim, H. Choi, and J. Park, "SDN-Assisted Slow HTTP DDoS Attack Defense Method", *IEEE Communications Letters*, vol. 22, no. 4, pp. 688–691, 2018.

[289] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "SENSS Against Volumetric DDoS Attacks", in *Proc. ACSAC*, San Juan, PR, USA, Dec. 2018.

[290] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of Intrusion Detection Systems: Techniques, Datasets and Challenges", *Cybersecur.*, Jul. 2019.

[291] J. Konstantas, *Enterprise Firewalls' Top Requirement: Scalability*, `https://www.securityweek.com/enterprise-firewalls-top-requirement-scalability`, Accessed: 2020-8-9, 2012.

[292] E. Damon, J. Mache, R. Weiss, *et al.*, "Chapter 31 - Cyber Security Education: The Merits of Firewall Exercises", in *Emerging Trends in ICT Security*, B. Akhgar and H. R. Arabnia, Eds., Boston: Morgan Kaufmann, 2014, pp. 507–516.

[293] Palo Alto Networks, *DoS Protection Profiles*, `https://bit.ly/3zsMRVW`, Accessed: 2020-09-09, 2020.

[294] Palo Alto Networks, *Reconnaissance Protection*, `https://bit.ly/3zo1A47`, Accessed: 2020-4-2, 2020.

[295] A. Sivanathan, H. Habibi Gharakheili, and V. Sivaraman, "Managing IoT Cyber-Security Using Programmable Telemetry and Machine Learning", *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 60–74, 2020.

[296] Impact Cyber Trust, *DARPA 2009 Intrusion Detection Dataset*, `http://www.darpa2009.netsec.colostate.edu/`, Accessed: 2020-8-9, 2020.

[297] Palo Alto Networks, *Flood Protection*, `https://bit.ly/3gEEAqj`, Accessed: 2020-09-09, 2020.

[298] M. Rahouti, K. Xiong, N. Ghani, and F. Shaikh, "SYNGuard: Dynamic Threshold-based SYN Flood Attack Detection and Mitigation in Software-Defined Networks", *IET Networks*, vol. 10, Jan. 2021.

[299] A. Hounsel, P. Schmitt, K. Borgolte, and N. Feamster, "Encryption without Centralization: Distributing DNS Queries across Recursive Resolvers", in *Proc. ANRW*, Virtual Event, USA, Jul. 2021.