Graph-Based Machine Learning for Passive Network Reconnaissance within Encrypted Networks

by

Kyle Alexander Millar

Bachelor of Engineering (Computer Systems Engineering, Honours),

University of Adelaide, 2017

Thesis submitted for the degree of

Doctor of Philosophy

in

School of Electrical & Electronic Engineering

Faculty of Science, Engineering and Technology

The University of Adelaide

September 2022

Supervisors:

- Prof. Cheng-Chew Lim[1]
- Dr. Hong Gunn Chew[1]
- Dr. Adriel Cheng[1,2]

[1]School of Electrical and Electronic Engineering, The University of Adelaide, Australia

[2]Information Sciences Division, Defence Science & Technology Group, Australia

Contents

Abstract

Network reconnaissance identifies a network's vulnerabilities to both prevent and mitigate the impact of cyber-attacks. The difficulty of performing adequate network reconnaissance has been exacerbated by the rising complexity of modern networks (e.g., encryption). We identify that the majority of network reconnaissance solutions proposed in literature are infeasible for widespread deployment in realistic modern networks.

This thesis provides novel network reconnaissance solutions to address the limitations of the existing conventional approaches proposed in literature. The existing approaches are limited by their reliance on large, heterogeneous feature sets making them difficult to deploy under realistic network conditions. In contrast, we devise a bipartite graph-based representation to create network reconnaissance solutions that rely only on a single feature (e.g., the Internet protocol (IP) address field). We exploit a widely available feature set to provide network reconnaissance solutions that are scalable, independent of encryption, and deployable across diverse Internet (TCP/IP) networks.

We design bipartite graph embeddings (BGE); a graph-based machine learning (ML) technique for extracting insight from the structural properties of the bipartite graph-based representation. BGE is the first known graph embedding technique designed explicitly for network reconnaissance. We validate the use of BGE through an evaluation of a university's enterprise network. BGE is shown to provide insight into crucial areas of network reconnaissance (e.g., device characterisation, service prediction, and network visualisation).

We design an extension of BGE to acquire insight within a private network. Private networks—such as a virtual private network (VPN)—have posed significant challenges for network reconnaissance as they deny direct visibility into their composition. Our extension of BGE provides the first known solution for inferring the composition of both the devices and applications acting behind diverse private networks.

This thesis provides novel graph-based ML techniques for two crucial aims of network reconnaissance—device characterisation and intrusion detection. The techniques developed within this thesis provide unique cybersecurity solutions to both prevent and mitigate the impact of cyber-attacks.

Citation Listing

This thesis is presented by publication. The contributing research papers are included *as is* within this thesis. The contributing papers are grouped into four chapters based on the overall theme of their content.

The order of the publications as they appear in the thesis is as follows:

Chapter 3	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, “Deep Learning for Classifying Malicious Network Traffic” in Trends and Applications in Knowledge Discovery and Data Mining, Cham, M. Ganji, L. Rashidi, B. C. M. Fung, and C. Wang, Eds., 2018, pp. 156-161.
	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, “Using Convolutional Neural Networks for Classifying Malicious Network Traffic” in Deep Learning Applications for Cyber Security, 2019, pp. 103-126.
Chapter 4	K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim, “Characterising Network-Connected Devices Using Affiliation Graphs,” in IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1-6.
	K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim, “Clustering Network-Connected Devices Using Affiliation Graphs,” in IEEE International Conference on Machine Learning and Cybernetics (ICMLC), 2021, pp. 1-6.
	A. Cheng and K. Millar, “Detecting Data Exflitration using Seeds based Graph Clustering,” accepted for publication in IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), 2022.

Chapter 4	K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim, “Operating System Classification: A Minimalist Approach,” in IEEE International Conference on Machine Learning and Cybernetics (ICMLC), 2020, pp. 143-150.
Chapter 5	K. Millar, L. Simpson, A. Cheng, H.G. Chew, and C.-C. Lim, “Detecting Botnet Victims Through Graph-Based Machine Learning,” in IEEE International Conference on Machine Learning and Cybernetics (ICMLC), 2021, pp. 1-6. K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim, “Enhancing Situational Awareness in Encrypted Networks Using Graph-Based Machine Learning,” submitted for publication in IEEE Transactions on Network and Service Management, 2022.
Chapter 6	K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim, “PiPiN: Acquiring Situational Awareness Behind Private Networks with Confidence,” submitted for publication in IEEE Transactions on Network and Service Management, 2022.

Declaration

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Kyle Alexander Millar	Date

Acknowledgment

I could not have completed this thesis without the unwavering support of those around me. It takes a village to do a task worth doing and I would be amiss to not recognise the people that have supported me throughout the most challenging endeavour I have embarked on.

I would first like to express my deepest gratitude to my supervisors: Prof. Cheng-Chew Lim, Dr. Hong Gunn Chew, and Dr. Adriel Cheng. Thank you for the countless hours you have given me in support, guidance, and encouragement. Thank you for challenging me to improve my research and for your patience in reviewing countless drafts. I would not have been able to complete this thesis without your tireless supervision and I will be forever indebted.

To my family, thank you for your ongoing love and support. In particular, for putting up with my chaotic work schedule over these past four years. I look forward to a healthier work/life balance in the future.

To my friends, those that came before and those that I made along the way, thank you for providing an outlet for my research. I will forever remember our Friday night catch ups and your constant encouragement over the years.

To Isobel, my partner and muse, thank you for sharing in the highs and lows of this chapter in my life. I could not have made it through the last push of writing without your constant encouragement. I look forward to sharing each and every chapter of my life with you.

I would also like to take this opportunity to formally recognise the staff at the University of Adelaide for whom this thesis would not be possible. In particular, I would like to thank the staff from the school of Electrical and Electronic Engineering (EEE), the graduate management centre, Information Technology and Digital Services (ITDS), and the high performance computing (HPC) team. In addition, I would like to thank the researchers at the Defence Science & Technology (DST) Group for providing advice and feedback, and the opportunity to pursue challenging and applicable research directions.

I formally acknowledge that the research contained within this thesis was supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

List of Figures

List of Tables

Symbols and Definitions

The symbols and definitions listed are in regard to the main body of text within this thesis. Symbols and definitions used in the contributing publications have been included as is and therefore may contain inconsistencies. Inconsistencies contained in the included publications have been detailed at the start of their corresponding chapter within this thesis.

List of Acronyms

Acronym	**Description**
ACSC	Australian Cyber Security Centre
ANU	Australian National University
AUD	Australian Dollar
BGE	Bipartite Graph Embeddings
BYO	Bring-Your-Own
C2	Command and Control
CDN	Content Delivery Network
CNN	Convolutional Neural Network
COVID-19	Novel Coronavirus Disease (2019)
DBN	Deep Belief Network
DNS	Domain Name System
DPI	Deep Packet Inspection
DST	Defence Science and Technology
FPR	False Positive Rate
GAN	Generative Adversarial Network
GBT	Gradient Boosting Tree
GCN	Graph Convolutional Neural Network

Acronym	**Description**
GNN	Graph Neural Network
HMM	Hidden Markov Model
HPC	High Performance Computing
HTTP	Hypertext Transport Protocol
IoT	Internet-of-Things
ISP	Internet Service Provider
IT	Information Technology
ITDS	Information Technology and Digital Services
kNN	k-Nearest Neighbours
MAC	Media Access Control
ML	Machine Learning
MLP	Multi-Layered Perceptron
NAT	Network Address Translation
NB	Naive Bayes
NIDS	Network Intrusion Detection Systems
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NN	Neural Network
OS	Operating System
OSINT	Open-Source INTelligence
P2P	Peer-to-Peer
PI	Prediction Interval

Acronym	**Description**
PII	Personal Identifiable Information
PiPiN	Private interval - Private Network
PoA	Point-of-Analysis
QoS	Quality of Service
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Network
RTP	Research Training Program
SSH	Secure Shell
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TCP/IP	Internet Protocol Suite (Transmission Control Protocol / Internet Protocol)
TES	Transient Edge Sampling
TTL	Time-to-Live
TTP	Tools, Techniques, and Procedures
UDP	User Datagram Protocol
UofA	University of Adelaide
URL	Uniform Request Locator
VPN	Virtual Private Network

Chapter 1

Introduction

The growing scale and complexity of modern communication networks has resulted in a central axiom of effective cybersecurity—"*You cannot secure what you cannot see*" [1, 2, 3]. This axiom highlights that often the main difficulty in providing effective cybersecurity is simply knowing what to protect.

Network reconnaissance is the key method of generating situational awareness within a network environment [4]. Network situational awareness[1] provides the necessary insight for a network operator or security analyst to investigate the behaviour of their network and its constituent devices. The insight gained through network reconnaissance not only allows for the identification of vulnerabilities that may exist on the network but also enables faster response times in the event of a cyber-attack [5].

The efficacy of current network reconnaissance techniques has decreased in recent years [6]. Current techniques were not envisioned to handle the complexity of modern networks and the pervasive use of encryption[2] [9]. To addresses these challenges, machine learning (ML) has been widely theorised for its application to network reconnaissance [10].

In this thesis, we devise novel ML-based techniques for network reconnaissance. In particular, we devise a graph-based representation of TCP/IP traffic to provide situational awareness when minimal prior information is available. We show that this graph-based

[1]The term *situational awareness* will be used exclusively to refer to *network situational awareness* for the remainder of this thesis.
[2]It is estimated that between 85%-95% of Internet communication is now encrypted [7, 8].

representation enables the development of novel network reconnaissance solutions that are scalable, independent of encryption, and deployable across diverse TCP/IP networks.

1.1 Motivation

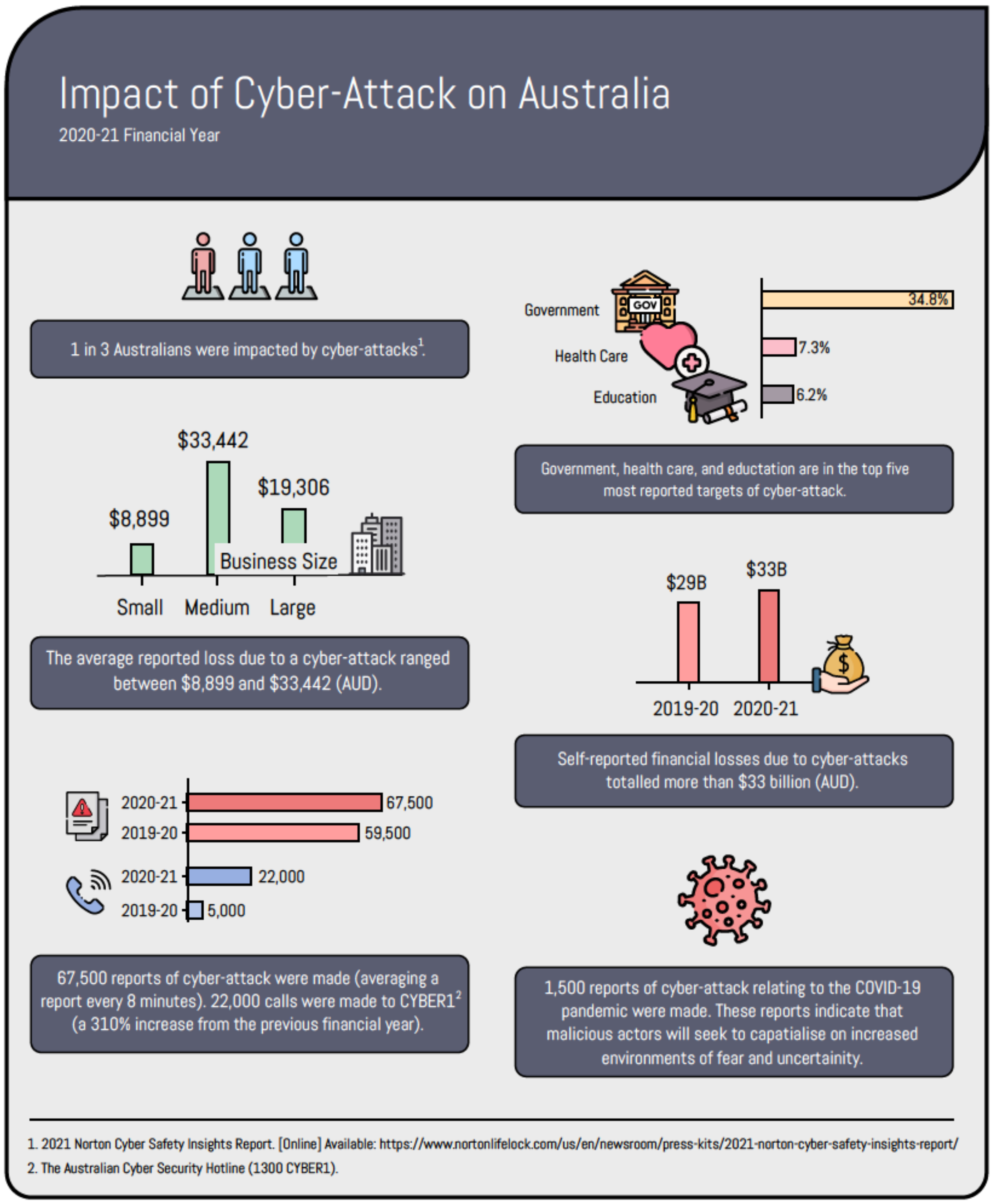
Recent years have marked an unprecedented increase in the volume and complexity of cyber-attacks worldwide [11]. Within Australia alone, a cyber-attack was reported every eight minutes during the 2020-21 financial year; incurring a reported loss of more than $33 billion (AUD) [12]. When polled in the same financial year, 62% of the Australian public feared cyber-attack as a critical threat to Australia's vital interests [13]; the highest percentage of all reported threats to Australia including climate change (61%) and the prevailing COVID-19 pandemic (59%).

A summary of the impact of cyber-attack on Australia during the 2020-21 financial year is illustrated in Figure 1.1. These statistics—provided by the Australian Cyber Security Centre (ACSC)—highlight the increasing rate and scale of cyber-attacks targeting Australia and its critical infrastructure (e.g., hospitals). It is evident that the rate of cyber-attacks will continue to grow as our reliance on the Internet increases. It is thus essential to explore novel preventative and mitigation strategies to stem the tide of online malicious activity.

Cybersecurity is the practice of protecting systems, networks, programs, and data from cyber-attack [15]. Implementing effective cybersecurity practices is thus essential to protect the services we interact with every day.

The National Institute of Standards and Technology (NIST) lists five fundamental functions of effective cybersecurity [16]: Identify, Protect, Detect, Respond, and Recover. Network reconnaissance is pivotal for achieving both the Identify and Detect functions:

- **Identify** provides insight into how a network operates, such as the number of devices on the network, how the devices are configured, and the assigned user roles.



Impact of Cyber-Attack on Australia

2020-21 Financial Year

1 in 3 Australians were impacted by cyber-attacks[1].

The average reported loss due to a cyber-attack ranged between $8,899 and $33,442 (AUD).

67,500 reports of cyber-attack were made (averaging a report every 8 minutes). 22,000 calls were made to CYBER1[2] (a 310% increase from the previous financial year).

Government, health care, and eductation are in the top five most reported targets of cyber-attack.

Self-reported financial losses due to cyber-attacks totalled more than $33 billion (AUD).

1,500 reports of cyber-attack relating to the COVID-19 pandemic were made. These reports indicate that malicious actors will seek to capatialise on increased environments of fear and uncertainity.

1. 2021 Norton Cyber Safety Insights Report. [Online] Available: https://www.nortonlifelock.com/us/en/newsroom/press-kits/2021-norton-cyber-safety-insights-report/
2. The Australian Cyber Security Hotline (1300 CYBER1).

Figure 1.1: The impact of cyber-attack on Australia (2020-21 financial year). Statistics sourced from ACSC's Annual Cyber Threat Report 2019-20 [14] and 2020-21 [12].

This insight provides the necessary knowledge to not only facilitate better quality of service (QoS) on a network but also to identify the network's vulnerabilities before they can be exploited.

- **Detect** enables the discovery of when a network has been compromised by a cyber-attack. Network reconnaissance can determine whether a network has been compromised by detecting such events as internal scanning[3], command-and-control (C2) communications[4]; and the exfiltration of data from the network. The prompt detection of malicious activity is essential for reducing its potential impact to the network and its users.

The consequences of inadequate network reconnaissance were illustrated in the 2018-19 cyber-attack of the Australian National University (ANU) [19]. ANU's inability to locate legacy and at-risk devices on their network was shown to significantly contribute to the exfiltration of up to 19-years' worth of sensitive data[5]. In remediation, ANU has committed to ongoing vulnerability assessments of their network. Conducting a vulnerability assessment of such a large network is however non-trivial. Improved network reconnaissance techniques have therefore become of key interest to industry, government, and academia alike.

Machine learning has become the catalyst for the development of novel network reconnaissance techniques [10]. The key benefit of ML is its ability to learn useful models from large, heterogeneous data. This benefit has enabled ML to achieve state-of-the-art results in a broad range of applications, such as image classification [20], natural language processing (NLP) [21], and autonomous vehicles [22]. ML is not however the panacea that it is often

[3]Scanning is a network reconnaissance technique that actively probes a network for vulnerabilities. Unsanctioned internal scanning is often an early indicator that a device has been compromised on the network [17].

[4]Command-and-control (C2) communications are the signals used by malicious actors to control the compromised devices on a network [18].

[5]The malicious actor(s) gained access to up 19-years' worth of human resources, finance, student administration, and e-forms data; however, it is believed that not all data was exfiltrated from ANU's network [19].

seen as; distinct application domains require distinct ML approaches [23]. In Chapters 2-3 we show that the majority of ML-based solutions for network reconnaissance utilise generic ML techniques that are often sub-optimal for the network reconnaissance domain.

In Chapters 4-6, we provide novel ML techniques designed specifically for the network reconnaissance domain. Our techniques enable the acquisition of situational awareness directly from a bipartite graph-based representation of a TCP/IP network. The bipartite representation is easily constructed, independent of encryption, and can be used to model the long-term behaviour of a TCP/IP network.

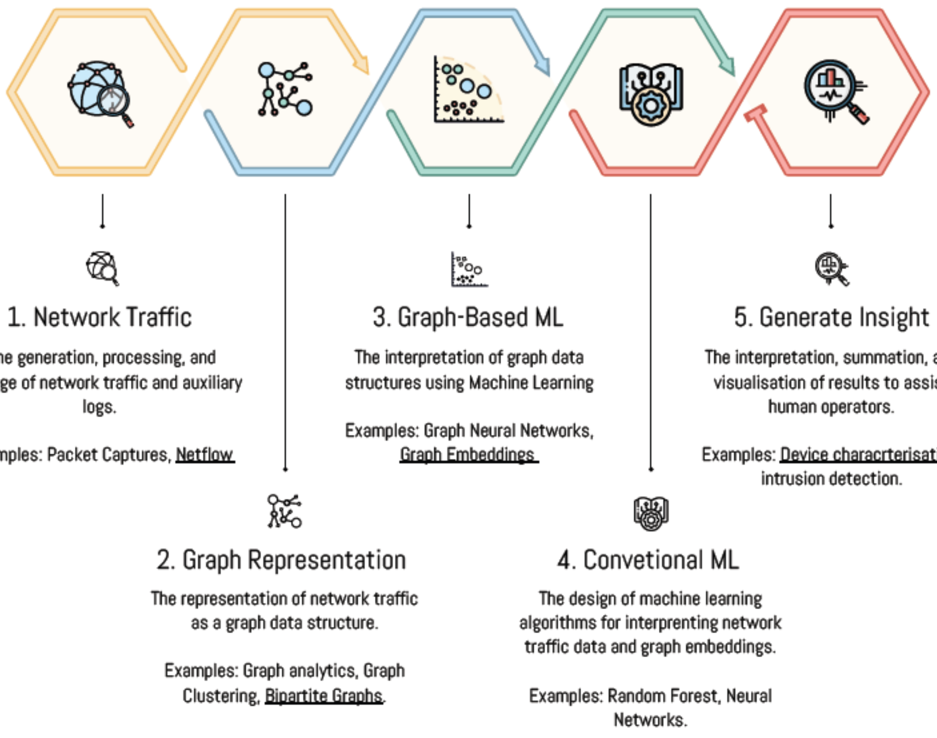
1.2 Aim and Scope

The aim of this thesis is to provide novel practical solutions for conducting network reconnaissance within Internet (TCP/IP) networks. The solutions developed within this thesis aim to assist network operators to better manage and secure the networks they administrate through increased situational awareness.

The application of network reconnaissance is divided into two principal methods [24]: 1) active network reconnaissance, where a network is stimulated to reveal its characteristics, and 2) passive network reconnaissance, where a network's characteristics are resolved through monitoring its pre-existing communication traffic[6]. This thesis focuses solely on providing novel passive network reconnaissance solutions.

Passive network reconnaissance was investigated as its use does not disturb the day-to-day operations of a network [25]. Furthermore, active network reconnaissance cannot be used to conduct a retrospective analysis of a network. This limitation of active network

[6] A detailed comparison of active and passive network reconnaissance is provided in Section 2.2.



1. Network Traffic

The generation, processing, and storage of network traffic and auxiliary logs.

Examples: Packet Captures, Netflow

2. Graph Representation

The representation of network traffic as a graph data structure.

Examples: Graph analytics, Graph Clustering, Bipartite Graphs.

3. Graph-Based ML

The interpretation of graph data structures using Machine Learning

Examples: Graph Neural Networks, Graph Embeddings

4. Convetional ML

The design of machine learning algorithms for interpreting network traffic data and graph embeddings.

Examples: Random Forest, Neural Networks.

5. Generate Insight

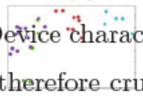
The interpretation, summation, and visualisation of results to assist human operators.

Examples: Device characrterisation, intrusion detection.

Figure 1.2: A high-level overview of the aim and scope of the thesis. The aim of the thesis is to provide effective network reconnaissance solutions. Machine learning (ML)—and subsequently graph-based ML—was shown to address key challenges of providing effective network reconnaissance in modern networks (Chapter 5).

reconnaissance would restrict the ability to investigate how a network was compromised in the event of a cyber-attack.

1. Network Traffic



This thesis focuses on two key objectives when conducting passive network reconnaissance—device characterisation and intrusion detection. Device characterisation allows for the detection of vulnerable devices on a network. It is therefore crucial for the development of preventative solutions for cybersecurity. Alternatively, intrusion detection allows for the detection of when a network has become the target of a cyber-attack. Effective intrusion detection solutions thus allow for quicker response times to mitigate the effect of an ongoing cyber-attack.

A high-level overview of the aim and scope of the thesis is provided in Figure 1.2. Conventional ML-based network reconnaissance solutions are introduced in Chapter 2 and evaluated in Chapter 3. In Chapter 4, we pose a novel bipartite graph-based representation of network traffic to address the limitations identified in the conventional ML approaches. In Chapters 5-6, we devise novel graph-based ML techniques for conducting network reconnaissance through the bipartite representation.

The graph-based ML techniques designed within this thesis utilise graph embeddings. Graph embeddings are an unsupervised ML technique that encode a graph's structural properties into a low-dimensional vector representation [26]. In this thesis, we show how graph embeddings address the four main challenges of performing network reconnaissance on modern networks (Section 1.3).

1.3 Challenges

The main challenges of network reconnaissance can be summarised by the four Vs of big data: Volume, Variability, Velocity, and Visibility; where:

Volume is the amount of data to be analysed. In network reconnaissance, data is derived from the network activity of the devices under analysis. The volume of such data has drastically increased within modern networks. For example, an enterprise network (such as a university's campus network) can easily reach sizes of tens to hundreds of thousands of devices. The sheer volume of the resultant network traffic generates a vast quantity of data to be analysed. The designed network reconnaissance solutions must minimise the required data to be processed to enable their feasible application to large-scale modern networks.

Variability is the diversity of data under analysis. In network reconnaissance, variability of data is derived from the diversity of devices, OSs, and applications, and monitoring processes used on the network under analysis. This diversity vastly exacerbates the difficultly of providing comprehensive network reconnaissance solutions that are applicable across diverse TCP/IP networks. The designed network reconnaissance solutions must be able to draw insight from data that is commonly generated by unique devices and software, and is commonly captured across diverse network monitoring solutions.

Velocity is determined by how quickly the data under analysis changes. Velocity is a key challenge in network reconnaissance due to the rate in which new devices, applications, and software updates are released to market. In each of these changes lies the potential to modify the underlying statistics used to characterise a network. The designed network reconnaissance solutions must therefore be easily modifiable in response to changes of the underlying network.

Visibility is the amount of data available for analysis. Encryption is the principal challenge in network reconnaissance visibility. Current techniques of network reconnaissance have become obsolete as the data they relied upon has been obfuscated by widespread encryption standards. Novel network reconnaissance solutions must therefore be developed to provide situation awareness even in the presence of encrypted network traffic (Chapter 4-5) and other anonymisation techniques such as private networks (Chapter 6).

In Chapter 2, we formalise the defined challenges of network reconnaissance through an analysis of 56 solutions proposed in related work. From this analysis, we identify four criteria that are required to enable the realistic application of network reconnaissance across diverse TCP/IP networks:

- Criterion 1 (CN1) - Encryption Independence
- Criterion 2 (CN2) - Universal Minimum Feature Set
- Criterion 3 (CN3) - Real-World Deployment
- Criterion 4 (CN4) - Long-Term Deployment

In Chapters 4-6, we develop the first network reconnaissance solutions that achieve all four criteria for realistic deployment across diverse TCP/IP networks.

1.4 Statement of Contribution

The solutions developed within this thesis provide network operators and security analysts with novel solutions for acquiring situational awareness within their networks. The provided solutions enable the characterisation of the devices on a network (e.g., identifying the OS or manufacturer of a device) and detecting potential areas of vulnerability and compromise.

The key novelty of the solutions developed within this thesis is the use of a graph theoretic approach to passive network reconnaissance. Through this approach, we prove that device characterisation and intrusion detection can be achieved through the exclusive analysis of the IP address field. We further the contribution of this approach through the use of graph-based ML; in which, we provide novel graph embedding solutions for passive network reconnaissance that are shown to be scalable, independent of encryption, and universally applicable to diverse TCP/IP networks.

The principal contribution of this thesis is four-fold:

1. **(Chapter 3)** We provide *Segmented-CNN*s; a novel CNN architecture designed to exploit the unique structural properties of the TCP/IP protocol stack. The segmented-CNN utilises a divide and conquer approach to evaluate the distinct properties of the header and payload sections of a TCP/IP packet. We show that this approach reduces the required training time of the classifier and improves robustness to evasive malicious behaviour. Furthermore, we discovered the proclivity of neural network architectures to overfit when evaluated on full packet captures. This discovery prompted the investigation into a graph-based point-of-analysis (PoA) for network reconnaissance.
2. **(Chapter 4)** We provide the first comprehensive framework for conducting passive network reconnaissance that relies only on the IP addresses field. The provided framework represents an enterprise network as a bipartite graph to exploit the

inherent community structure in the Internet services used by the devices on a network. Through the sole reliance on the IP address field; the proposed framework is scalable, independent of encryption, and widely deployable across diverse TCP/IP networks. We demonstrate the realistic application of the framework on captured network data from The University of Adelaide (UofA). We provide evidence that supports the use of the framework for device characterisation (e.g., operating system classification) and intrusion detection (e.g., detecting data exfiltration and botnet infrastructure).

3. **(Chapter 5)** We design bipartite graph embeddings (BGE). BGE is the first graph embedding technique that enables the real-time analysis of a large enterprise network. We show that BGE remains effective under partial network observation and efficiently scales for the analysis of networks containing hundreds of thousands of devices. We provide BGE as a packaged tool[7] that can be used to generate insight into any bipartite graph structure (e.g., recommendation and citation networks).

4. **(Chapter 6**) We validate the use of prediction intervals (PIs) and ensemble models to provide a confidence metric for ML-based network reconnaissance solutions. This confidence metric quantifies the model and data uncertainty of a neural network to inform a network operator as to whether the predictions made can be accepted or require further investigation.

   We provide PiPiN; an extension of BGE that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. PiPiN is the first known solution for inferring the composition of both the devices and applications acting behind diverse private networks. We show that PiPiN can accurately estimate the composition of device manufacturers, operating systems, and applications that are used within a private network.

[7] `https://github.com/MillarK-UofA/bipartite_graph_embeddings`

1.5 Overview

This thesis is comprised of seven chapters. It is presented by publication; for which, Chapters 3-6 are comprised of nine papers on the topic of passive network reconnaissance. The included papers have been combined into selected chapters based on the similarity of their research aims and contributions. Figure 1.3 provides an overview of the thesis' flow and chapter structure.

In summary, the thesis is structured as follows:

In Chapter 2, an extensive literature review of network reconnaissance is provided. This literature review introduces the current state-of-the-art network reconnaissance techniques and further defines the contribution gap to be addressed in this thesis. Furthermore, supplementary background knowledge on network reconnaissance is provided within this chapter.

In Chapter 3, we evaluate the use of conventional (ML) techniques for conducting network reconnaissance. In particular, we provide an empirical justification for the prevalent use of RF and CNNs that was identified in related work (Chapter 2). Furthermore, we discover the proclivity of neural network architectures to overfit when evaluated on full packet captures. This discovery prompted the investigation of a graph-based representation for network reconnaissance.

In Chapter 4, a novel representation of an enterprise network as a bipartite graph is proposed and evaluated. The bipartite graph representation is evaluated manually, through conventional graph theoretic approaches, and using conventional ML techniques. The identified limitations of these evaluations motivated the investigation of graph-based ML in the remaining chapters.

In Chapter 5, we provide two novel graph-based (ML) techniques for conducting passive network reconnaissance. These techniques were created to address the key limitations identified in Chapter 4. In particular, we designed bipartite graph embeddings (BGE);

the first graph embedding technique that enables the real-time analysis of a large enterprise network. We show that the embeddings produced by BGE can be reused to satisfy distinct network reconnaissance objectives (e.g., device and application characterisation). BGE thus provides a comprehensive methodology for acquiring situational awareness within a TCP/IP network.

In Chapter 6, we provide the first utilisation of graph embeddings for private network analysis. Private networks—such as a virtual private network (VPN)—have posed significant challenges for network reconnaissance as they deny direct visibility into their composition. Graph embeddings were shown to be able to reveal key insights into the overall behaviour of the private network; such as the composition of manufacturers, operating systems, and applications used on the network.

In Chapter 7, the thesis is concluded with a summary of the work conducted within the thesis and suggestions for future work in the network reconnaissance domain.

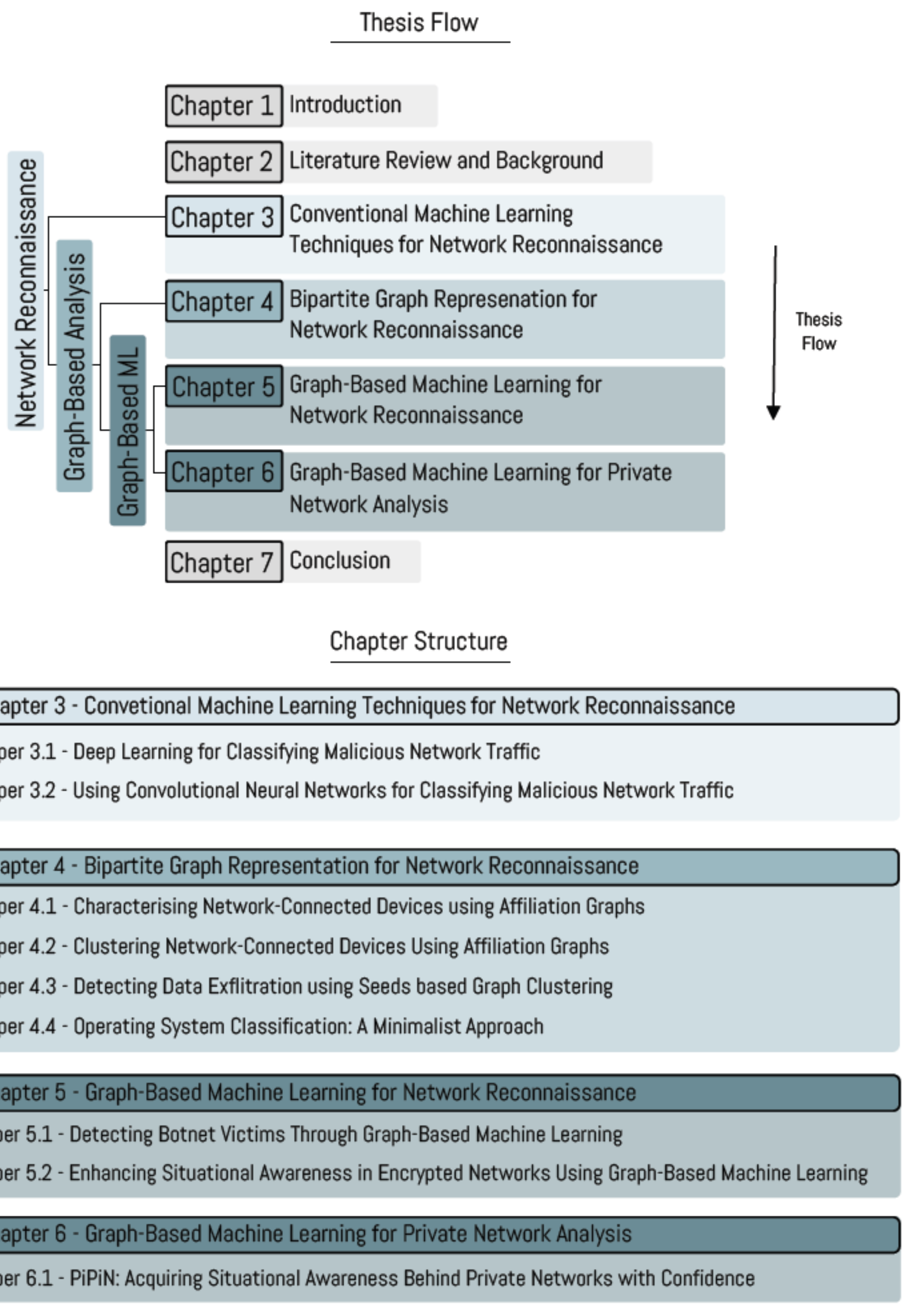

Thesis Flow
Chapter 1 Introduction
Chapter 2 Literature Review and Background
Chapter 3 Conventional Machine Learning Techniques for Network Reconnaissance
Chapter 4 Bipartite Graph Represenation for Network Reconnaissance
Chapter 5 Graph-Based Machine Learning for Network Reconnaissance
Chapter 6 Graph-Based Machine Learning for Private Network Analysis
Chapter 7 Conclusion
Network Reconnaissance
Graph-Based Analysis
Graph-Based ML
Thesis Flow
Chapter Structure
Chapter 3 - Convetional Machine Learning Techniques for Network Reconnaissance
Paper 3.1 - Deep Learning for Classifying Malicious Network Traffic
Paper 3.2 - Using Convolutional Neural Networks for Classifying Malicious Network Traffic
Chapter 4 - Bipartite Graph Representation for Network Reconnaissance
Paper 4.1 - Characterising Network-Connected Devices using Affiliation Graphs
Paper 4.2 - Clustering Network-Connected Devices Using Affiliation Graphs
Paper 4.3 - Detecting Data Exflitration using Seeds based Graph Clustering
Paper 4.4 - Operating System Classification: A Minimalist Approach
Chapter 5 - Graph-Based Machine Learning for Network Reconnaissance
Paper 5.1 - Detecting Botnet Victims Through Graph-Based Machine Learning
Paper 5.2 - Enhancing Situational Awareness in Encrypted Networks Using Graph-Based Machine Learning
Chapter 6 - Graph-Based Machine Learning for Private Network Analysis
Paper 6.1 - PiPiN: Acquiring Situational Awareness Behind Private Networks with Confidence

Figure 1.3: Overview of the thesis' flow and chapter structure.

Chapter 2

Literature Review and Background

In this chapter, we survey 56 papers in the network reconnaissance domain and provide a taxonomy of their motivation, methodology, and limitations. In addition, this chapter provides the reader with the necessary background knowledge on network reconnaissance required for the remainder of this thesis.

Chapter 2 is structured as follows. In Section 2.1, we discuss the common objectives of network reconnaissance solutions. In Section 2.2, we define active and passive network reconnaissance and compare their utilisation. This thesis focuses only on passive network reconnaissance. In Section 2.3, we provide an overview of passive network reconnaissance for acquiring situational awareness within an Internet (TCP/IP) network. Section 2.4 enumerates four criteria for the creation of feasible network reconnaissance solutions for real-world deployment. A summary of the related work in passive network reconnaissance is provided in Section 2.5 and the identified contribution gaps addressed in this thesis are outlined in Section 2.6.

2.1 Network Reconnaissance Objectives

The fundamental objective of network reconnaissance is to provide situational awareness within a network environment. Situational awareness, however, is multifaceted. Network operators often have distinct objectives when acquiring situational awareness within the networks that they administer.

We first enumerate the common objectives of network reconnaissance as found in related work. This thesis is focused on the use of network reconnaissance to provide situational awareness of the devices, applications, and users within a network environment. Related work investigating the characterisation of the network topology itself (e.g., network topology discovery [27]) are not considered. We taxonomise network reconnaissance solutions by four primary objectives: intrusion detection, device characterisation, application characterisation, and user identification. These four objectives are defined as follows:

Intrusion detection enables the discovery of compromises within a network. The primary objective of network intrusion detection systems (NIDS) is to allow for quicker response times in the event of a cyber-attack. Their secondary objective is to provide insight for preventative cybersecurity solutions to predict and mitigate future cyber-attacks.

NIDS are used to detect either anomalous or known signatures of malicious network behaviour [28]. The benefit of detecting anomalous network behaviour is that it can be used to detect novel cyber-attacks within a network. Novel cyber-attacks—often referred to as *zero-day* attacks [29]—target unknown vulnerabilities within a network and therefore cannot be detected by methods that rely on known signatures of malicious network behaviour. Detecting anomalous network behaviour may produce false positives[1]. False positives put a greater strain on the network operators as they are required to investigate each anomaly [30].

[1]A false positive is the incorrect prediction that a benign behaviour is malicious.

There are two key streams of research within the intrusion detection objective. First, machine learning (ML) is applied to automate the process of identifying signatures of known malicious behaviour [31, 32, 33]. This stream would allow signatures to be updated more frequently and provide quicker response times in the event of a cyber-attack. Second, network reconnaissance solutions are being investigated to reduce the false positive rate (FPR) when detecting anomalous behaviour [28, 30, 34]; alleviating the time-intensive task of tracking down wrongful identifications of malicious activity.

Device characterisation aims to identify the characteristics of the devices operating within a network. Identifying a device's characteristics—such as its operating system (OS) or manufacturer—is crucial for identifying its vulnerabilities [35]. For example, the EternalBlue exploit—infamously used in the WannaCry ransomware attack—exploited vulnerabilities within the Windows OS [36]. To secure a network, it is therefore essential to identify the characteristics—and hence the vulnerabilities—of its constituent devices.

Device characterisation enables more effective network management by providing insight into how a network is being utilised [37]. For example, a network composed of a singular type of device (e.g., all devices running the Windows OS) is more likely to be a computer lab rather than a bring-your-own (BYO) network composed of the personal devices brought onto the network by staff, students, or guests. These network compositions would require vastly distinct security and managerial requirements. For example, BYO networks may need stricter security policies as a personal device may already be in a compromised state unbeknownst to the user [38].

There are two main streams of research in the device characterisation objective. First, novel solutions are being developed to overcome the current challenges of device characterisation (e.g., encryption and the variety and volume of modern devices) [39, 40, 41, 42]. Second, techniques to infer device information within private networks (e.g., VPNs) are being investigated [43, 44, 45, 46]. Techniques to perform device characterisation within a private network enables network operators to have greater insight into the networks that they administer.

Application characterisation aims to identify the applications used by the devices operating within a network. Similar to device characterisation, application characterisation provides insight into a network's utilisation and vulnerabilities. This insight is essential for maintaining effective security and managerial policies. For example, an ISP may prioritise the network traffic associated with certain applications (such as video conferencing) to improve the quality of service (QoS) on the network. In addition, application characterisation is used to identify the vulnerabilities introduced into the network through its utilised applications. For example, exploits introduced into SolarWind's Orion application resulted in the compromise of over 18,000 networks; including those owned by Intel, Microsoft, and US government agencies [47].

There are two main streams of research in the application characterisation objective. The first stream focuses on the development of novel solutions to overcome the complexity of characterising application network behaviour [48, 49, 50, 51]. This complexity has arisen from the myriad of applications available, their widespread use of encryption, and their volatility (e.g., frequent updates). The second stream aims to utilise network reconnaissance to identify the actions a user takes within an application [52]. For example, identifying whether a user is sending a text message through WhatsApp or conversing through a video call. The ability to identify specific actions within an application has been used to facilitate auxiliary research into user identification.

User identification aims to identify and predict the behaviour of the users of a network. User identification is commonly investigated to identify the privacy risks caused by a user's Internet behaviour. For example, Stevens *et al.* [53] investigated thirteen advertisement (Ad) libraries commonly installed in Android applications. The authors found that such Ad libraries could be used to identify a specific user and track their location. It is critical that this leakage of personal identifiable information (PII) is detected and mitigated as this information can be used to extort, track, and harm the respective user.

There are two principal streams of research in user identification. The first stream is user fingerprinting [54, 55]. User fingerprinting utilises network reconnaissance to identify specific users on a network. This research is typically used for authentication and anomaly detection purposes. The second stream of research aims to improve user privacy protections through the detection and subsequent reduction of PII leakage [56, 53].

Figure 2.1 provides an overview of the four common objectives of passive network reconnaissance. This thesis provides solutions to address challenges within the intrusion detection, device characterisation, and application characterisation objectives. User identification was not directly investigated within this thesis; however, there exists potential for extending our developed techniques for use within the user identification objective. For example, Kurtz *et al.* [55] showed that personal configurations of mobile phones (e.g., installed apps) can be used to identify a specific mobile device and by extension its user. This finding posits that user identification can be conducted through addressing challenges within the application characterisation objective.

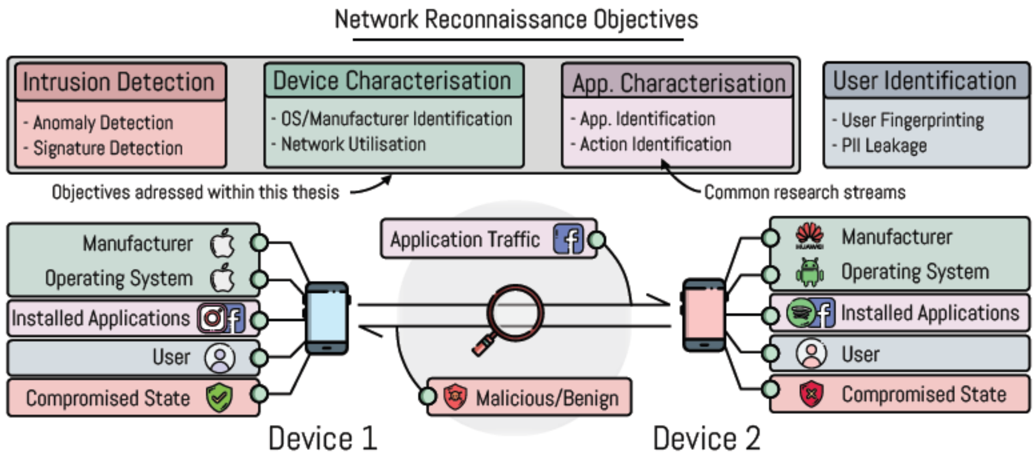


Figure 2.1: An illustration of four common objectives within the network reconnaissance domain. Further provided is an example of the potential insights that can be acquired within each objective. This thesis is focused on the intrusion detection, device characterisation, and application (app.) characterisation objectives.

2.2 Active versus Passive Network Reconnaissance

Network reconnaissance is divided into two principal methods [24]: 1) active techniques, for which a network is stimulated to reveal its characteristics; and 2) passive techniques, for which a network's characteristics are resolved through monitoring its pre-existing communication traffic. Figure 2.2 illustrates the key distinction between an active versus passive approach for conducting network reconnaissance.

Active network reconnaissance directly interrogates the network under analysis. This interrogation is conducted through the use of network *probes*. Probes are specially crafted network communications which stimulate the receiving device into revealing its characteristics [24]. The complexity of these probes ranges from trivial, such as identifying whether a device will simply respond to a probe; to exceedingly intricate, such as detecting the OS of a device through its combined response to 15 carefully crafted probes [57].

The key benefit of active reconnaissance is that the information gained is dictated by its operator. For example, a network operator that wishes to know the OS of a particular device can send out the 15 probes required for that knowledge. In contrast, a passive approach must wait for device of interest to reveal that same information. The trade-off of an active approach is its intrusiveness into a network.

Active reconnaissance directly interacts with the network it is monitoring. This interaction may be inappropriate when the use of the network is separate from its ownership [58]. For example, an internet service provider (ISP) may not wish to actively probe the devices of its customers. This action would be seen unfavourably by its customers and may cause the ISP to be liable if any disruption is caused by the probes they send. Furthermore, if the active reconnaissance of a network was unwarranted by its ownership, it could be mitigated through security services such as firewalls [59]. These services can be configured to either block incoming probes or to create false responses to obfuscate the reconnaissance.

The fundamental limitation of active network reconnaissance is its inability to perform a retrospective analysis of a network. That is, a device, application, or user can only be investigated if they are directly connected to the network. This limitation could cause severe restrictions when investigating how a network was compromised in the event of a cyber-attack. For example, if the device or user of interest is no longer connected to the network. Additionally, mobile devices—such as smartphones and laptops—are often only connected to the network for a limited period of time. This mobility further exacerbates the challenges faced in active network reconnaissance.

This thesis focuses solely on the use of passive network reconnaissance due to the limitations discussed within this section. Active and passive network reconnaissance techniques are not mutually exclusive. Optimal results could be achieved through deploying both active and passive approaches where applicable [59]. In this thesis, however, we focus on the advancement of passive network reconnaissance solutions given their wider application.

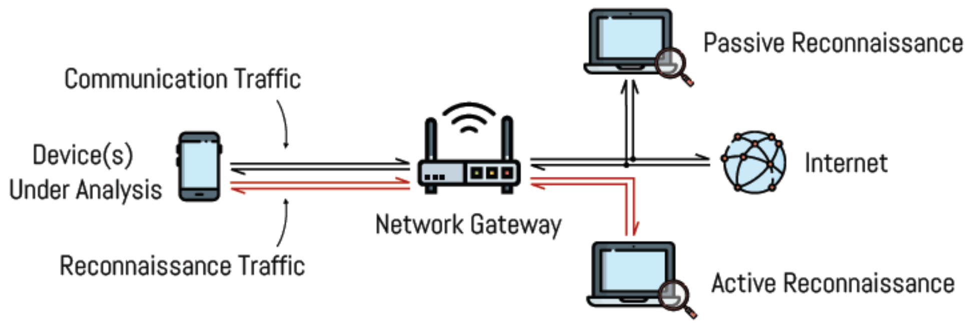


Figure 2.2: An illustration of the distinction between active and passive network reconnaissance. Active network reconnaissance is seen to directly interact with the network under analysis, whereas passive network reconnaissance relies on observing pre-existing network (communication) traffic.

2.3 Passive Network Reconnaissance

Passive network reconnaissance aims to characterise a network through monitoring its pre-existing communication traffic. The ability to acquire pertinent communication traffic is therefore tantamount to the insight that can be achieved through passive network reconnaissance. This section provides an overview of Internet (TCP/IP) networking in the context of the insight that can be achieved through passive network reconnaissance.

A message sent through a TCP/IP network is segmented into individual *packets* of data [60]. Each packet contains a subset of the message content (i.e., the packet's *payload*) as well as additional features (i.e., *metadata*) that the packet requires to traverse the network.

A packet in a TCP/IP network is encapsulated into five layers[2]: Application, Transport, Network, Link, and Physical [60]. Each layer provides a distinct role in forwarding a message through the network and ensuring that the message can be interpreted by its intended recipient.

The five layers of the TCP/IP protocol stack are depicted in Figure 2.3. Each layer provides unique insight when conducting passive network reconnaissance. For example, J. Martin *et al.* [61] predict the manufacturer and model of a device via its medium access control (MAC) address (*link layer*), whereas the time-to-live field (*network layer*) is commonly used to predict a device's OS [40, 62, 42]. The process of conducting passive network reconnaissance through the metadata found in specific layers of a TCP/IP packet is termed *packet-inspection*.

Packet inspection draws insight from the diverse layers in the TCP/IP protocol stack. A technique that utilises all TCP/IP layers is termed *deep packet inspection* (DPI). The key assumption of DPI is that the packet's payload (i.e., message content) is unencrypted

[2]This thesis conforms to the five-layer TCP/IP protocol stack framework as encouraged by Kurose and Ross [60]. However, other frameworks exist such as the seven-layer OSI model [60].

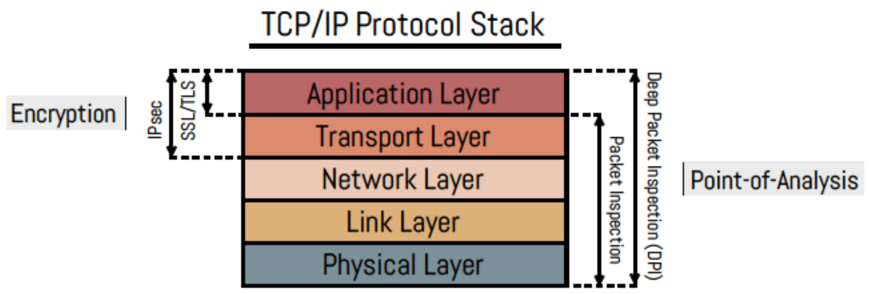


Figure 2.3: The five layer TCP/IP protocol stack. Layers have been highlighted that are either 1) obfuscated by common encryption standards; or, 2) observed in either packet inspection or deep packet inspection (Point-of-Analysis).

and observable[3]. The validity of this assumption has diminished in recent years due to the universal adoption of encryption standards [63]. Novel passive network reconnaissance solutions are now typically designed to acquire insight from either 1) lower layers of the TCP/IP protocol stack (i.e., packet inspection); or, 2) aggregated packet information to perform a flow-based analysis.

A TCP/IP network flow is defined as a set of packets passing an observation point that share the same flow key[4] [64]. The flow key is used to identify packets that are affiliated within the same communication. Analysis of the overall communication between two devices can reveal additional insight into their behaviour. For example, to detect the malicious exfiltration of data from a network, it is often essential to investigate the accumulative data sent through its communications rather than the data contained in each packet individually [65].

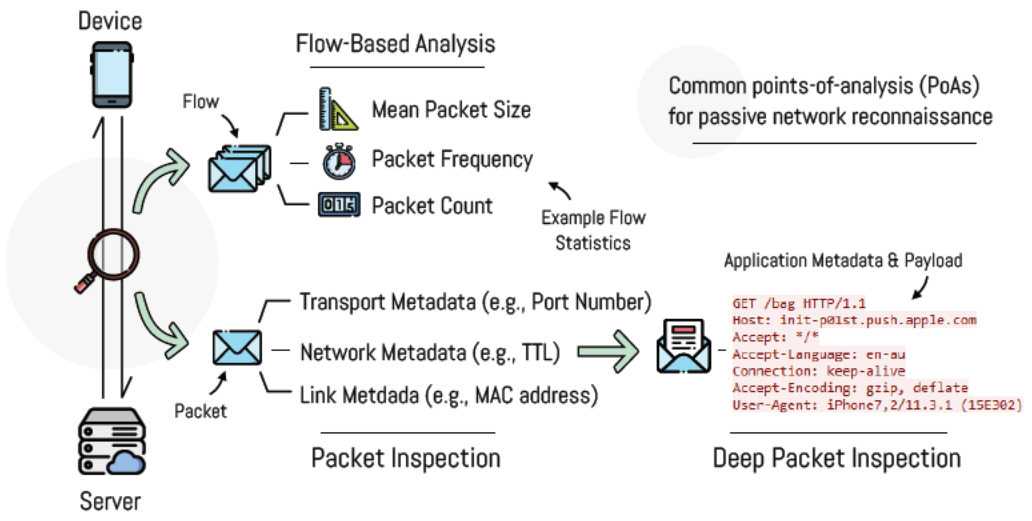
[3]The term deep packet inspection is commonly used to refer to all packet inspection techniques. This thesis utilises the defined distinction between packet inspection and deep packet inspection to highlight network reconnaissance solutions that rely on the explicit message content of a TCP/IP packet (i.e., the application layer).

[4]There are several definitions of what packet properties compose its flow key. In this thesis we conform to the most common flow key—the 5-tuple. The 5-tuple flow key is composed of the source and destination IP address, source and destination port number, and the transport protocol (e.g., TCP/UDP).

Flow-based analysis has been a key factor in recent network reconnaissance solutions. Two factors have led to the rise in flow-based network reconnaissance: 1) to reclaim the situational awareness that was lost due to widespread encryption [66]; and, 2) the use of machine learning to mitigate the challenge of identifying characteristic behaviour from flow-based statistics [67, 48, 54].

Figure 2.4 provides a summary of the common points-of-analysis (PoAs) used within passive network reconnaissance. While these PoAs are shown as separate, they are often used in conjunction to reveal further insight into the network. Within this thesis, we develop passive network reconnaissance solutions that utilise each PoA and their combination.

In Chapter 4, we pose a novel PoA based on a bipartite graph-based representation of a TCP/IP network. We validate our proposed bipartite PoA for intrusion detection, device characterisation, and application characterisation. We show that our bipartite PoA provides a unique method for conducting passive network reconnaissance that is scalable, independent of encryption, and deployable across diverse TCP/IP networks.


Device
Flow-Based Analysis
Common points-of-analysis (PoAs) for passive network reconnaissance
Flow
Mean Packet Size
Packet Frequency
Packet Count
Example Flow Statistics
Application Metadata & Payload
Transport Metadata (e.g., Port Number)
Network Metadata (e.g., TTL)
Link Metdada (e.g., MAC address)
Packet
Packet Inspection
Deep Packet Inspection
Server

```
GET /bag HTTP/1.1
Host: init-p01st.push.apple.com
Accept: */*
Accept-Language: en-au
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: iPhone7,2/11.3.1 (15E302)
```

Figure 2.4: An illustration summarising the common points of analysis (PoAs) for passive network reconnaissance.

2.4 Criteria for Feasible Network Reconnaissance Solutions

We identify four criteria from the survey of related work that measure the feasibility of a network reconnaissance solution for deployment on a real-world TCP/IP network:

- Criterion 1 (CN1) - Encryption Independence
- Criterion 2 (CN2) - Universal Minimum Feature Set
- Criterion 3 (CN3) - Real-World Deployment
- Criterion 4 (CN4) - Long-Term Deployment

2.4.1 Criterion 1 (CN1) - Encryption Independence

Network reconnaissance solutions that rely on unencrypted TCP/IP traffic have limited applicability. These solutions are restricted to either 1) the analysis of unencrypted communications (representing less than 15% of total Internet traffic [7, 8]); or, 2) assume the traffic can be decrypted before analysis restricting their applicability to tightly controlled network environments (e.g., utilising an SSL/TLS proxy server [68]).

Criterion 1 (CN1) validates the applicability of a proposed network reconnaissance solution on encrypted TCP/IP traffic. There are two main encryption standards used within TCP/IP networks—SSL/TLS and IPsec [69]. SSL/TLS provides application layer encryption, whereas IPsec encrypts both the application and transport layers [70]. A solution is deemed to meet the requirements of CN1 if it is applicable on both SSL/TLS and IPsec encryption standards.

Recent network reconnaissance solutions have posed the use of ML to perform DPI on encrypted TCP/IP network traffic. The proposed solutions conclude that the use deep neural networks (e.g., CNNs and RNNs) can identify latent signatures in encrypted packet content. The validity of the proposed solutions is disputable as they are not shown to generalise to unforeseen encrypted traffic[5]. In Chapter 3, we provide an evaluation of CNNs for DPI. We identify that the proposed solutions for DPI on encrypted TCP/IP traffic could simply be learning biases in the evaluated dataset. We note the uncertainty of such techniques when evaluated on CN1 to reflect this finding.

2.4.2 Criterion 2 (CN2) - Universal Minimum Feature Set

Network reconnaissance solutions are dependent on the availability of their respective feature set. This dependency poses a significant challenge as feature availability is subject to the network under analysis. For example, operational networks may restrict access to full packet captures (*pcaps*) to preserve the privacy and security of the network's users. This restriction would thus prevent the use of network reconnaissance solutions that are reliant on packet inspection.

Criterion 2 (CN2) validates the applicability of a proposed network reconnaissance solution for deployment across diverse TCP/IP networks. A solution is deemed to meet the requirements of CN2 if the solution enables network reconnaissance to be performed on any TCP/IP network without modifying the network's administration (e.g., requesting that additional network features be made available for analysis).

A flow-based PoA is the most widely used network monitoring method [71]. It is proven for high-speed networks and reduces the privacy constraints that arise from packet inspection. Furthermore, a flow-based PoA can be derived given access to the associated pcap files. For this reason, we consider a flow-based PoA when constructing a universal feature set that would be applicable across diverse TCP/IP networks.

[5]All surveyed solutions were evaluated on the same dataset(s) for both training and testing.

The primary flow-based monitoring standards are NetFlow v9 [72] and IPFIX [73]. A total of 104 and 491 possible flow statistics can be monitored by the respective standards. A small subset of the total flow statistics will be chosen when monitoring a network to reduce unnecessary overhead. For example, the flow-based monitoring solution deployed at the University of Adelaide utilises 25 (24%) of the possible NetFlow v9 flow statistics.

The flow statistics used to monitor a network are heuristically chosen by the network's operator(s). This heuristic feature selection process often limits the widespread applicability of network reconnaissance solutions as the flow statistics available in one network are not guaranteed to be available in another. A set of flow statistics that are common to all TCP/IP networks must therefore be defined to enable universal network reconnaissance solutions.

Table 2.1 enumerates the smallest set of flow-statistics that would be commonly available across diverse TCP/IP networks. These nine flow-statistics are widely monitored across TCP/IP networks as they represent the smallest set of flow-statistics that are required to adequately describe a flow [71]. We designated this set of flow-statistics as the universal minimum feature set.

Table 2.1: Universal Minimum Feature Set

#	Feature Name (IPFIX)	Description
1	flowStartMilliseconds	Timestamp of the flow's first packet
2	flowEndMilliseconds	Timestamp of the flow's last packet
3	sourceIPv4Address	IPv4 source address in the packet header
4	destinationIPv4Address	IPv4 destination address in the packet header
5	sourceTransportPort	Source port in the transport header
6	destinationTransportPort	Destination port in the transport header
7	protocolIdentifier	IP protocol number in the packet header
8	packetDeltaCount	Number of packets for the flow
9	octetDeltaCount	Number of bytes for the flow

The universal minimum feature set is used in this thesis to validate each surveyed solution for widespread applicability across diverse TCP/IP networks. A solution is deemed to meet the requirements of CN2 if it utilises only the set of universal minimum features for network reconnaissance as provided in Table 2.1.

2.4.3 Criterion 3 (CN3) - Real-World Evaluation

A proposed network reconnaissance solution must be evaluated under real-world network conditions to verify its feasibility. Synthetic data (e.g., ISCXVPN2016 [74]) or network captures from a controlled lab environment are suitable for prototyping purposes; however, they do not provide sufficient evidence that the proposed solutions are feasible under real-world network conditions.

Criterion 3 (CN3) assess whether a proposed network reconnaissance solution has been verified through a real-world evaluation. A solution is deemed to meet the requirements of CN3 if it has been evaluated on network captures taken from a real network that is being utilised in an operational context (e.g., a university's enterprise network). Real-time operation was not considered to be a major factor of a solution's suitability for real-world evaluation as it is highly dependent on the solution's operational environment (e.g., the size of the network under analysis).

The evaluation of CN3 is to identify network reconnaissance solutions that have been verified within realistic network conditions. A solution that does not satisfy the requirements of CN3 may still be applicable under realistic conditions, however, the solution has yet to be verified.

2.4.4 Criterion 4 (CN4) - Long-Term Deployment

TCP/IP networks are highly volatile. Network characteristics are modified due to software updates, changes in user behaviour, or through physical changes to the network itself.

Network reconnaissance solutions must therefore be able to update in response to changes in the underlying network.

Criterion 4 (CN4) verifies whether a proposed network reconnaissance solution has been assessed to provide long-term situational awareness on a network. A solution is deemed to meet the requirements of CN4 if either one of two conditions are met:

1. The solution has been shown to remain effective for at least one month after the solution was initially evaluated. A period of one month was chosen as it verifies that the technique is not simply biased to the initial time of its evaluation. This condition does not guarantee the length of time a network reconnaissance solution will remain effective. It is assumed that all network reconnaissance solutions will eventually need to be updated due to changing network behaviour.
2. The solution can be easily updated in response to changes in the network's behaviour. For example, S. Dong *et al.* [50] proposed a semi-supervised ML approach to enable their network reconnaissance solution to be continuously updated. The proposed semi-supervised approach allows their solution to be updated based on labelled and unlabelled network activity. Updating a solution based purely on labelled network activity—as required by a supervised ML solution—would be restrictive as labelling network data is difficult to perform at scale.

2.5 Summary of Related Work

This section evaluates 56 passive network reconnaissance solutions that have been proposed in related work. We taxonomise the proposed solutions by 1) their objective, 2) point-of-analysis (PoA), 3) their use of machine learning (ML), and 4) whether the solutions satisfy the four defined criteria (CN1-4) for widespread deployment on realistic network conditions.

A summary of the passive network reconnaissance solutions is provided in Table 2.2 (page 42). From the provided summary, we identify eight trends that unify the passive network solutions that have been proposed within the last decade.

2.5.1 Machine Learning

A timeline of all surveyed papers and their use of ML is provided in Figure 2.5. The overwhelming majority (88%) of passive network reconnaissance solutions proposed in literature have utilised ML. This trend is a result of 1) the overall increase in the multi-disciplinary application of ML; and 2) the challenges of big data (i.e., volume, variety, velocity, and visibility) that have been further introduced into passive network reconnaissance.

It is evident that ML is pivotal to the advancement of passive network reconnaissance solutions on modern networks. A consensus, however, has not been reached on which ML techniques are best applied for network reconnaissance. Figure 2.6 categorises the ML techniques used in the surveyed solutions. In total, 23 distinct ML techniques were utilised for passive network reconnaissance within the surveyed papers.

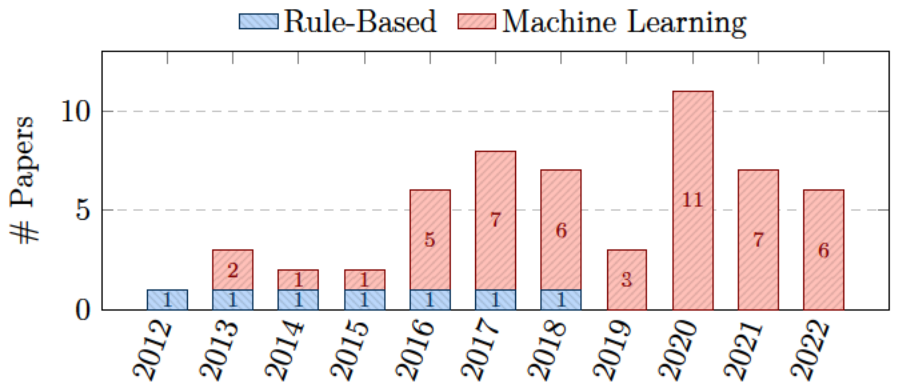


Figure 2.5: Timeline of surveyed papers and their use of machine learning. All surveyed network reconnaissance solutions that were published after 2017 have utilised ML.

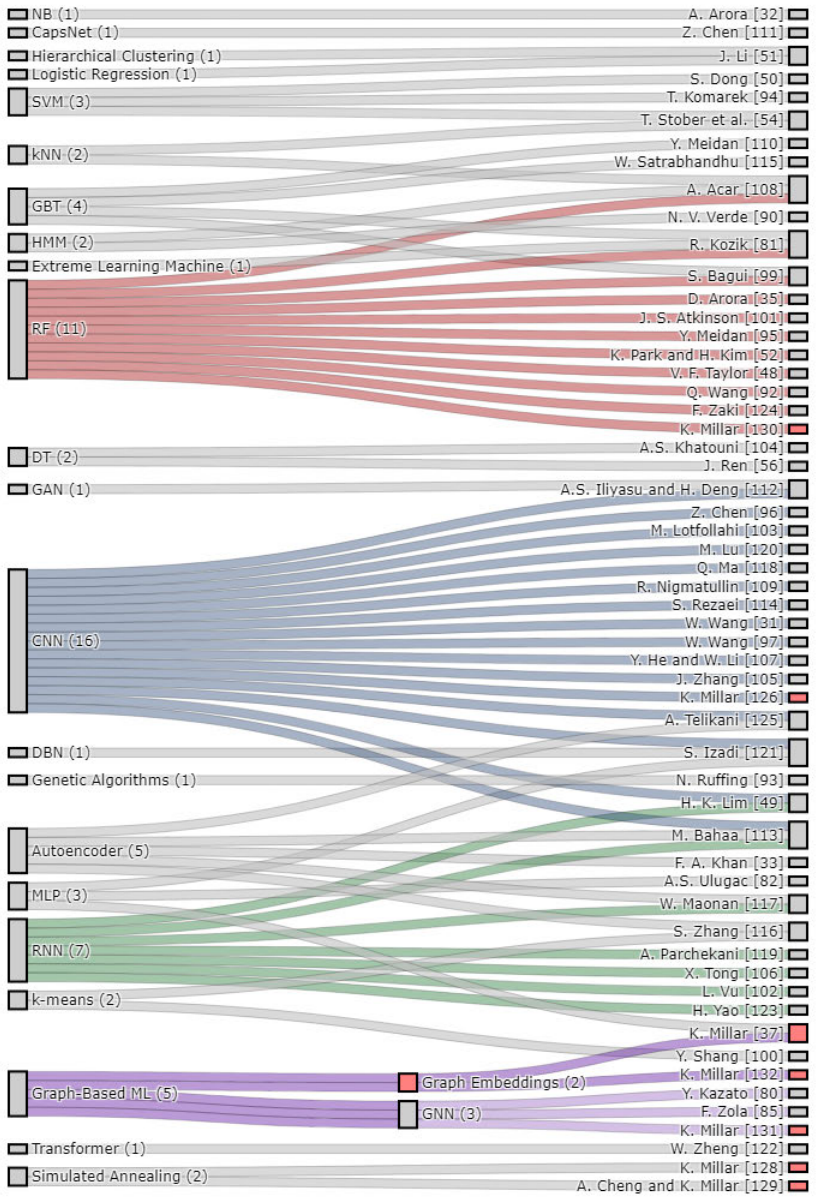


Figure 2.6: A snakey diagram illustrating the machine learning techniques (*left*) used for passive network reconnaissance within the surveyed papers (*right*). Random Forest (RF), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) are the most widely utilised ML-based techniques for network reconnaissance. The papers provided within this thesis have been included to highlight their contribution in the passive network reconnaissance domain (*red boxes*).

The two most utilised ML-based techniques for passive network reconnaissance are 1) flow-based analysis utilising random forest (RF) and 2) sequential flow/packet analysis utilising convolutional neural networks (CNNs) or recurrent neural networks (RNNs). In Chapter 3, we provide an analysis of why RF and CNNs have become pervasive ML-based techniques in the network reconnaissance domain.

2.5.2 Objectives

The underlying objectives of passive network reconnaissance have largely been unaltered over the past decade. This finding highlights that novel solutions must be developed to address the evolving challenges of conducting network reconnaissance within modern networks.

A timeline of the network reconnaissance objectives investigated over the past decade is provided in Figure 2.7. It is evident that application characterisation has become a central focus for network reconnaissance over the past five years. The motivation for recent studies of application characterisation can be summarised into three factors:

1. The reduced efficacy of port-based analysis. The transport layer's pre-assigned port numbers have historically been used to characterise TCP/IP applications (e.g., HTTP: 80, SSH: 22, DNS: 53) [75]. Port-based analysis has largely become obsolete due to dynamic port allocation and the funnelling of distinct applications through a single port (e.g., HTTPS: 443) [63, 76].
2. The increase in widespread encryption. Application characterisation through DPI was introduced to address the decline in efficacy of port-based analysis (e.g., nDPI [77]). The widespread utilisation of encryption, however, has reduced the effectiveness of DPI for application characterisation.
3. The increase in distinct application traffic. A significant factor in the rise of distinct application traffic has been caused by the proliferation of mobile applications (e.g.,

Facebook, Instagram, and TikTok). The Google play store alone contains over 2.6 million distinct applications [78]. This rapid influx in heterogeneous applications has vastly increased the complexity of application characterisation.

These three factors are not limited to just application characterisation. For example, device characterisation has experienced a rapid influx of heterogeneous devices due to the introduction of the internet-of-things (IoT) [79]. To contrast the large body of work on application characterisation, this thesis provides solutions to address intrusion detection and device characterisation[6].

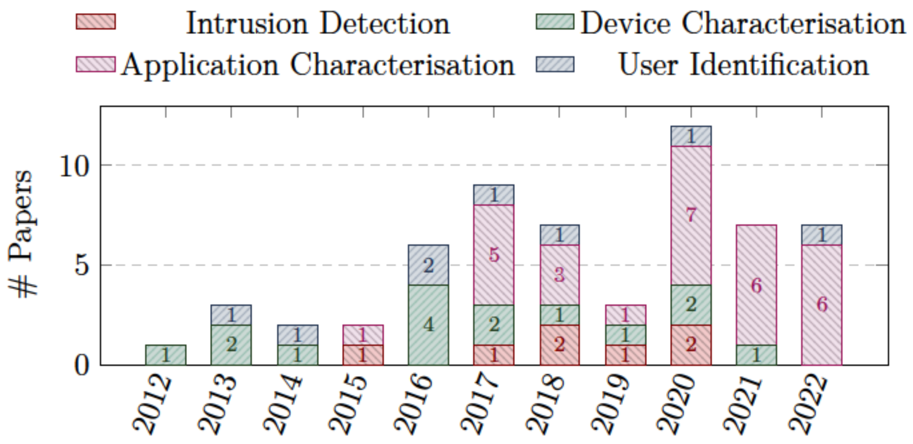


Figure 2.7: Timeline of surveyed papers and their objective when conducting passive network reconnaissance.

[6]In Chapters 5-6, we evaluate our devised network reconnaissance solutions on both device and application characterisation.

2.5.3 Criterion 1 (CN1) - Encryption Independence

Encryption was the central factor in the development of passive network reconnaissance solutions over the past decade. 93% of surveyed solutions assumed that encryption would be applied to at least the application layer (i.e., SSL/TLS encryption), whereas 84% assumed that both the application and transport layer would be encrypted (i.e., IPsec).

A total of 17 papers conclude that the use of either CNNs or RNNs can be used for DPI on encrypted TCP/IP traffic. This claim would require overwhelming evidence as it requires information retrieval from encrypted packet content.

Packet data would result in a distinct cipher text when encrypted with different keys[7]. It must be assumed that the proposed solutions for encrypted DPI are learning from either 1) unencrypted TCP/IP layers within the packet content; or 2) synthetic biases within the evaluated dataset. In Chapter 3, we provide evidence to suggest that such solutions are simply overfitting to synthetic biases within their respective dataset.

2.5.4 Criterion 2 (CN2) - Minimum Universal Feature Set

Only 2 (3.6%) of the surveyed papers satisfied the requirements of CN2. This finding highlights a clear absence in network reconnaissance solutions that would be widely applicable across diverse TCP/IP network configurations. For instance, 34 (61%) of the proposed solutions rely on packet-inspection. Packet-inspection could not be applied for widespread deployment as access to packet captures is highly restricted due to the privacy and security concerns they introduce.

[7]TCP/IP traffic is encrypted using block ciphers [60]. There is no predetermined mapping between the plain text (i.e., unencrypted data) and cipher text (i.e., encrypted data) in a block cipher without knowing the associated key.

The two solutions that met the requirements of CN2 are as follows:

1. Kazato *et al.* [80] utilised the source and destination IP address fields for intrusion detection. The authors propose a solution to automate the detection of malicious services hosted on the Internet. Their proposed solution examines an Internet service's IP address, domain name, and uniform request locator (URL) to identify whether the service is malicious. Open source intelligence (OSINT) is used to resolve the domain name and URL of a service given its IP address. The authors show that insightful network reconnaissance can be conducted with access to only the IP address of a service.
2. Kozik *et al.* [81] provided a solution for intrusion detection that utilises six out of the nine features in the universal minimum feature set: source/destination IP address, source/destination port number, number of bytes, and number of packets. Their proposed solution utilised a flow-based analysis to detect botnet behaviour. Overall, the solution achieves a high detection rate of botnet behaviour. The authors thus show that intrusion detection can be performed using only the set of universal minimum features.

Both Kazato *et al.* [80] and Kozik *et al.* [81] focused on the intrusion detection objective. In Chapters 4-6 we provide network reconnaissance solutions that satisfy the requirements of CN2 that can be used for intrusion detection, device characterisation, and application characterisation.

2.5.5 Criterion 3 (CN3) - Real World Analysis

A total of 14 (24%) of the surveyed papers satisfied the requirements of CN3. Of these 14 papers: ten evaluated their technique on an operational network (e.g., an enterprise or internet service provider (ISP) network); two papers evaluated their technique through a user study; and the remaining two papers evaluated their technique through captures of real-world malicious activity.

The evaluation of a proposed solution on an operational network provides sufficient bounds for the feasibility of the solution within realistic deployments [82]. This finding is supported by the high number of solutions evaluated within operational networks. To adhere to this convention, we evaluate the techniques provided within this thesis on a real operational network—provided by the University of Adelaide—where applicable[8].

2.5.6 Criterion 4 (CN4) - Long-Term Deployment

A total of 12 (21%) of the surveyed papers met the requirements of CN4. Of these 12 papers: ten evaluated the performance of their proposed solution for at least one month after its initial analysis; and two papers provided a mechanism to update their solution in response to changes in the network environment.

The following two papers provide a method to update their solution in response to changes in the network environment:

1. S. Dong *et al.* [50] proposed a semi-supervised support vector machine (SVM) for flow-based application characterisation. Their solution identifies unlabelled network flows that are close to the SVM's decision boundary—that is, flows that cannot be confidently classified. The flows that cannot be confidently classified are manually reviewed and used to update the SVM. This approach minimises the number of labelled flows that are required to update their proposed solution.

[8]The operational network that was evaluated within this thesis did not contain any known samples of malicious behaviour. For intrusion detection, we evaluated our proposed techniques on widely utilised datasets containing examples of malicious behaviour (UNSW-NB15 [83] and ISCXBot2014 [84]). These datasets are synthetic, short-term captures and thus do not meet the requirements of CN3 or CN4.

2. J. Li *et al.* [51] proposed an open-world assumption for flow-based application characterisation. The authors' open-world assumption allows for the characterisation of applications that were not present when the solution was initially trained. This open-world assumption highlights a significant limitation in conventional supervised ML solutions—they are only effective if the same set of classes (e.g., applications) are present when both training and testing. This limitation would reduce the effectiveness of supervised ML solutions on TCP/IP networks due to the volatility of such networks (e.g., the rate in which applications, devices, and software updates are released to market).

The benefits of both approaches can be achieved through the utilisation of unsupervised ML. First, an unsupervised ML technique could be continuously trained as it does not require labelled training data[9]. Second, continuous training of an unsupervised ML technique inherently provides an open-world assumption as the training is not restricted by *a priori* knowledge (i.e., labelled classes).

We validate the techniques provided within this thesis on captures of an operational network taken over a six-month period. Furthermore, we devise an unsupervised ML technique (Chapter 5, Paper 5.2) to provide a network reconnaissance solution that can be easily updated in response to changes in the network environment.

2.5.7 Point-of-Analysis

There have been noticeable changes in the points-of-analysis (PoAs) used in passive network reconnaissance solutions as illustrated in Figure 2.8. Two key developments were identified. First, there has been a resurgence of DPI due to the use of sequential ML tech-

[9]Domain knowledge or a small sample of labelled data would be required to interpret the output of an unsupervised ML technique.

niques (i.e., CNNs and RNNs); and second, a graph-based PoA[10] for conducting passive network reconnaissance has been introduced.

The graph-based PoA has been introduced to address the limitations in flow-based analysis. That is, the classification of individual flow records is insufficient to capture the long-term characteristics of a specific host [81]. In Chapter 4, we devise a bipartite graph-based PoA to create network reconnaissance solutions that rely only on a single feature (e.g., the Internet protocol (IP) address field). We exploit a universally available feature set to provide network reconnaissance solutions that are scalable, independent of encryption (CN1), and deployable across diverse Internet (TCP/IP) networks (CN2).

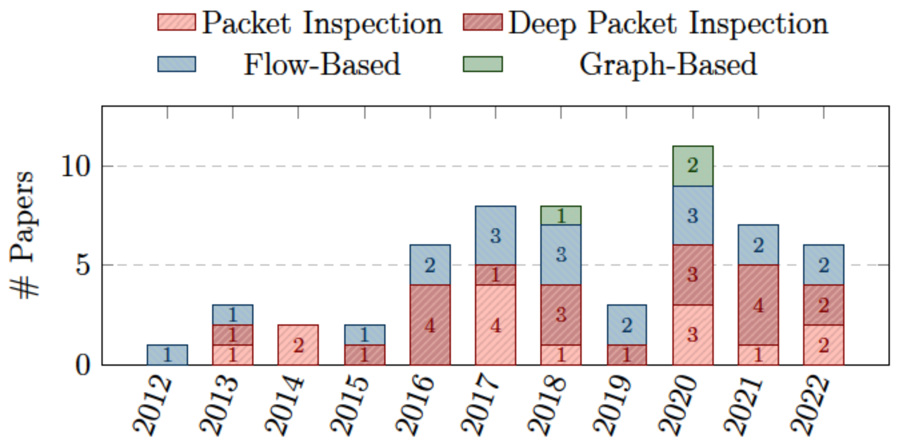


Figure 2.8: Timeline of surveyed papers and their point-of-analysis (PoA).

[10]A graph-based PoA for network analysis is not novel. Its use, however, has largely been confined to network topology analysis. The solutions surveyed in this thesis utilise a graph-based PoA for host-based characterisation on a TCP/IP network.

2.5.8 Graph-Based Machine Learning

Three surveyed papers performed graph-based network reconnaissance. Of which, two papers utilised graph-based ML and one utilised conventional ML to analyse the structural properties of a graph-based PoA. We focus on the graph-based ML solutions as they are tailored for the analysis of graph data structures[11].

There are two main approaches for graph-based ML:

1. **Graph embeddings** encode a graph's structural properties into a low-dimensional vector representation [26]. The benefit of graph embeddings is that they provide a universal representation that can be reused for different application domains. Furthermore, graph embeddings rely only on the structural properties of a graph and are therefore an unsupervised ML technique.
2. **Graph characterisation** analyses a graph's structural properties to perform analysis for a specific application domain. Labelled training data (i.e., supervised ML) is used to tailor the analysis for the intended application domain.

Zola *et al.* [85] and Kazato *et al.* [80] both propose a graph-based ML solution for network reconnaissance through graph characterisation. Furthermore, both proposed solutions utilised a graph convolutional neural network (GCN) as defined by Kipf and Welling [86]. This definition proposes a multi-layered neural network that propagates the structural and nodal attributes of a graph through each layer of the neural network. The simplified form of a Kipf and Welling's propagation rule is as follows:

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}) \tag{2.1}$$

[11]The limitations of applying conventional ML for the analysis of graph data structures is provided in Chapter 5.

where $H^{(l)}$ is the l-th layer in the neural network, σ is a non-linear activation function, A is the graph's adjacency matrix, and $W^{(l)}$ is the weight matrix at the l-th in the neural network.

Equation 2.1 states that the graph's adjacency matrix—that is, all connections within the graph—must be known in order to propagate information through the GCN. This definition of a GCN is an example of transductive learning. Transductive learning requires the entire graph to be present when training and testing; thus limiting such solutions to singular, static graphs [87].

A TCP/IP network is neither singular nor static. A transductive GCN would be sub-optimal for network reconnaissance as it would require retraining for each analysis conducted. To mitigate this limitation, we pose the first use of graph embeddings for passive network reconnaissance.

Graph embeddings preserve a graph's structural properties within a d-dimensional latent space (Equation 2.2) [88].

$$\theta : v \rightarrow \vec{x} \in \mathbb{R}^d \text{ for all } v \in V \tag{2.2}$$

where θ is a structure preserving mapping function from vertex, v, to the vertex's latent space representation, $\vec{x}$; d is the dimensionality of the latent space; and V is the set of all vertices within the graph.

Graph embeddings were chosen for the development of novel network reconnaissance solutions for the following three key reasons:

1. **Inductive learning:** Graph embeddings provide a framework for conducting graph analytics without requiring access to the entire graph. This property is essential for the analysis of a TCP/IP network as it allows for the analysis of highly dynamic and large-scale networks.

2. **Objective independent embeddings:** Graph embeddings are an unsupervised ML technique and thus are not biased to a particular objective. This property allows for a universal representation of a TCP/IP network that can be used to meet distinct network reconnaissance objectives. In Chapters 5-6, we show that the same embeddings can be utilised for both device and application characterisation.

3. **Long-Term Deployment (CN4):** Graph embeddings are a form of unsupervised ML and therefore can provide an open-world assumption and are easily updated in response to changes on the underlying network. Graph embeddings are therefore well-suited to support the long-term deployment of network reconnaissance solutions.

In Chapter 5, we design a novel method of deriving the mapping function, θ, for use within bipartite graphs. In particular, we develop a novel neural network architecture and training schema to exploit the unique structure of dynamic, large-scale bipartite graphs.

2.6 Contribution Gap

Three principal contribution gaps were identified through the summary of related work that was outlined in Section 2.5:

1. Machine learning: We show that there has been no consensus on which ML techniques should be utilised for passive network reconnaissance. In particular, the renewed investigation of deep learning techniques in related fields (e.g., image classification) has introduced a plethora of novel approaches for conducting passive network reconnaissance. In Chapter 3, we compare the use of machine and deep learning techniques for conducting network reconnaissance for both DPI and flow-based analysis. We identify severe limitations of such techniques as summarised by the four criteria for realistic deployment (CN1-4).

2. Criteria for realistic deployment: No surveyed solution met all four criteria for realistic deployment. In particular, only two (3.6%) of the proposed solutions are independent of encryption (CN1) and relied only on a universal minimum feature set (CN2). In chapter 4, we devise a bipartite graph-based PoA to provide a universal framework for conducting passive network reconnaissance that satisfies both the requirements of CN1 and CN2. We show that our provided framework allows for achieving diverse objectives within passive network reconnaissance (e.g., intrusion detection, device characterisation, and application characterisation).

Only 8 (14%) of the surveyed papers evaluated their proposed solution for long-term deployment (CN4) in real-world networks (CN3). To meet these criteria, we validate our bipartite graph-based representation using captures from a university campus network taken over six months. In Chapters 4-6, we develop the first passive network reconnaissance solutions that achieve all four criteria for realistic deployment (CN1-4).

3. Graph-based machine learning: All known graph-based ML solutions for passive network reconnaissance have utilised graph characterisation. In contrast, Chapter 5 provides a graph embedding technique design specifically for passive network reconnaissance. Graph embeddings were investigated as they do not require labelled data to be trained (i.e., unsupervised ML). This property allows graph embeddings to be easily updated in response to changes in the network environment (CN4). Furthermore, we show that graph embeddings can be used to achieve diverse network reconnaissance objectives without requiring explicit re-training for each objective.

Table 2.2: A summary of related work in passive network reconnaissance.

		Objectives				Point-of-Analysis (PoA)					Criteria				
											CN1				
Paper	Year	Intrusion Detection	Device Characterisation	Application Characterisation	User Identification	Packet Inspection	Deep Packet Inspection	Flow-Based	Graph-Based	Machine Learning	SSL/TLS	IPsec	CN2	CN3	CN4
[43]	2012	–	✓	–	–	–	–	✓	–	–	✓	×	×	✓	✓
[54]	2013	–	–	–	✓	–	–	✓	–	✓	✓	✓	×	×	×
[89]		–	✓	–	–	✓	✓	–	–	–	×	×	×	×	✓
[82]		–	✓	–	–	✓	–	–	–	✓	✓	✓	×	✓	×
[40]	2014	–	✓	–	–	✓	–	–	–	–	✓	×	×	✓	✓
[90]		–	–	–	✓	–	–	✓	–	✓	✓	✓	×	✓	✓
[91]	2015	✓	–	–	–	–	–	✓	–	–	✓	✓	×	×	×
[92]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×
[93]	2016	–	✓	–	–	✓	–	–	–	✓	✓	✓	×	×	×
[52]		–	–	–	✓	✓	–	–	–	✓	✓	✓	×	×	×
[35]		–	✓	–	–	–	–	✓	–	✓	✓	✓	×	✓	×
[61]		–	✓	–	–	✓	–	–	–	–	✓	✓	×	✓	✓
[56]		–	–	–	✓	✓	✓	–	–	✓	×	×	×	✓	✓
[94]		–	✓	–	–	✓	✓	–	–	✓	×	×	×	✓	×
[95]	2017	–	✓	–	–	–	–	✓	–	✓	✓	✓	×	×	✓
[96]		–	–	✓	–	✓	–	–	–	✓	✓	✓	×	×	×
[31]		✓	–	–	–	✓	✓	–	–	✓	?	?	×	✓	✓
[97]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[98]		–	✓	–	✓	✓	–	–	–	–	✓	×	×	×	×
[32]		✓	–	–	–	–	–	✓	–	✓	✓	✓	×	×	×
[99]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×

Table 2.2: A summary of related work in passive network reconnaissance (*cont.*).

Paper	Year	Objectives				Point-of-Analysis (PoA)				Machine Learning	Criteria				
											CN1				
		Intrusion Detection	Device Characterisation	Application Characterisation	User Identification	Packet Inspection	Deep Packet Inspection	Flow-Based	Graph-Based		SSL/TLS	IPsec	CN2	CN3	CN4
[100]	2018	✓	–	–	–	–	–	✓	✓	✓	✓	✓	×	×	×
[101]		–	–	–	✓	✓	–	–	–	✓	✓	✓	×	×	×
[48]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	✓
[81]		✓	–	–	–	–	–	✓	–	✓	✓	✓	✓	×	×
[102]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[103]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[42]		–	✓	–	–	✓	✓	–	–	–	×	×	×	✓	×
[49]	2019	–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[33]		✓	–	–	–	–	–	✓	–	✓	✓	✓	×	×	×
[104]		–	✓	–	–	–	–	✓	–	✓	✓	×	×	×	×
[105]	2020	–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[106]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[107]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[108]		–	✓	–	✓	✓	–	–	–	✓	✓	✓	×	×	×
[109]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×
[110]		–	✓	–	–	–	–	✓	–	✓	✓	✓	×	×	✓
[111]		–	–	✓	–	✓	–	–	–	✓	✓	✓	×	×	×
[112]		–	–	✓	–	✓	–	–	–	✓	✓	✓	×	×	×
[80]		✓	–	–	–	–	–	–	✓	✓	✓	✓	✓	✓	×
[113]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[114]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	✓	×

Table 2.2: A summary of related work in passive network reconnaissance (*cont.*).

		Objectives				Point-of-Analysis (PoA)					Criteria				
											CN1				
Paper	Year	Intrusion Detection	Device Characterisation	Application Characterisation	User Identification	Packet Inspection	Deep Packet Inspection	Flow-Based	Graph-Based	Machine Learning	SSL/TLS	IPsec	CN2	CN3	CN4
[115]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×
[116]		–	✓	–	–	✓	–	–	–	✓	✓	×	×	×	×
[50]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	✓	✓
[117]	2021	–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[118]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[119]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[120]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[51]		–	–	✓	✓	✓	–	–	–	✓	✓	✓	×	✓	✓
[121]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×
[122]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[123]	2022	–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[124]		–	–	✓	–	–	–	✓	–	✓	✓	✓	×	×	×
[125]		–	–	✓	–	✓	✓	–	–	✓	?	?	×	×	×
[85]		✓	–	–	–	–	–	–	✓	✓	✓	✓	×	×	×
Techniques developed within this thesis.															
Paper 3.2 [126]		✓	–	–	–	✓	✓	–	–	✓	?	?	×	×	×
Paper 4.1 [127]		–	✓	–	–	–	–	–	✓	–	✓	✓	✓	✓	×
Paper 4.2 [128]		–	✓	–	–	–	–	–	✓	✓	✓	✓	✓	✓	×
Paper 4.3 [129]		✓	–	–	–	–	–	–	✓	✓	✓	✓	✓	✓	×
Paper 4.4 [130]		–	✓	–	–	–	–	–	✓	✓	✓	✓	✓	✓	✓
Paper 5.1 [131]		✓	–	–	–	–	–	–	✓	✓	✓	✓	×	×	×
Paper 5.2 [132]		–	✓	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓
Paper 6.1 [37]		–	✓	✓	–	–	–	–	✓	✓	✓	✓	✓	✓	✓

Chapter 3

Conventional Machine Learning Techniques for Network Reconnaissance

This chapter evaluates the use of conventional machine learning (ML) techniques for conducting network reconnaissance. Furthermore, we present a novel convolutional neural network (CNN) architecture to explore how a conventional ML technique can be tailored for the network reconnaissance domain.

Two papers are presented within this chapter:

Paper 3.1 - "*Deep Learning for Classifying Malicious Network Traffic*", compares the use of five prevalent ML techniques for intrusion detection: random forest (RF), support vector machines (SVMs), decision trees, multi-layered perceptron (MLP), and CNNs. Furthermore, a comparison of deep packet inspection (DPI) and flow-based analysis is provided.

Paper 3.2 - "*Using Convolutional Neural Networks for Classifying Malicious Network Traffic*", provides a novel CNN architecture—*Segmented-CNN*—that is tailored for the network reconnaissance domain. The Segmented-CNN architecture exploits the structural properties of the TCP/IP protocol stack to reduce the classifier's training time and improve its resilience to evasive malicious behaviour.

The key contribution of this chapter is as follows:

1. **(Paper 3.1)** We provide an empirical justification for the prevalent use of RF and CNNs that was identified in related work (Section 2.5.1). RF and CNN techniques were shown to exhibit the highest performance for flow-based and DPI network reconnaissance solutions, respectively.
2. **(Paper 3.2)** We design *Flow-Image*; a novel representation of network traffic for use within a CNN classifier. The Flow-Image is the first image representation that preserves the categorical features of network data. This insight has been utilised in subsequent publications applying image processing techniques to network data [113].
3. **(Paper 3.2)** We provide *Segmented-CNN*s; a novel CNN architecture designed to exploit the unique structural properties of the TCP/IP protocol stack. The Segmented-CNN utilises a divide and conquer approach to evaluate the distinct properties of the header and payload sections of a TCP/IP packet. We show that this approach reduces the required training time of a CNN classifier and improves robustness to evasive malicious behaviour.
4. **(Paper 3.2)** We expose the proclivity of neural network architectures to overfit the training set when conducting DPI. It was shown that a packet's identification field and checksum were the two most significant features when using either MLP or CNNs. It is evident that these architectures are overfitting the training set as these two features are both unique to their individual packet. This contribution illustrates the fallacy in using ML for DPI. The vast quantity of information contained in packet content is readily overfit. It is thus critical to investigate ML-based techniques for network reconnaissance that are less reliant on exhaustive feature sets.

Statement of Authorship

Title of Paper	Deep Learning for Classifying Malicious Network Traffic
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, A. Cheng, H. G. Chew, and C. C. Lim, "Deep Learning for Classifying Malicious Network Traffic," in Trends and Applications in Knowledge Discovery and Data Mining, Cham, M. Ganji, L. Rashidi, B. C. M. Fung, and C. Wang, Eds., 2018: Springer International Publishing, pp. 156-161.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Deep Learning for Classifying Malicious Network Traffic[⋆]

K. Millar, A. Cheng, H.G. Chew, and C.-C. Lim

School of Electrical and Electronic Engineering, University of Adelaide, Australia
{kyle.millar, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au

Abstract. As the sophistication of cyber malicious attacks increase, so too must the techniques used to detect and classify such malicious traffic in these networks. Deep learning has been deployed in many application domains as it is able to learn patterns from large feature sets. Given that the implementation of deep learning for network traffic classification is only just starting to emerge, the question of how best to utilise and represent network data to such a classifier still remains. This paper addresses this question by devising and evaluating three different ways of representing data to a deep neural network in the context of malicious traffic classification. We show that although deep learning does not show significant improvement over other machine learning techniques using metadata features, its use on payload data highlights the potential for deep learning to be incorporated into novel deep packet inspection techniques. Furthermore, we show that useful predictions of malicious classes can still be made when the input is limited to just the first 50 bytes of a packet's payload.

Keywords: Deep learning · convolutional neural networks · Internet traffic classification · malicious traffic detection.

1 Introduction

As the number of users who rely on the Internet in their professional and personal lives increases, so too does the profit in exploiting the vulnerabilities of these networking systems. Recent years have shown an unprecedented number of malicious attacks worldwide. By design, malicious attacks are not easily identifiable, with each passing year producing new ways to foil existing systems of detection. Therefore, in today's constantly evolving networks, there arises the need for a classification system capable of adapting to these changes.

Deep learning has become highly prominent in the machine learning community due to the availability of big data and the specialised hardware required to utilise it. Deep learning's advantage lies in its ability to learn and adapt to

⋆ This research is supported by the Commonwealth of Australia as represented by the Defence Science and Technology Group of the Department of Defence. The authors acknowledge the following for their contributions in gathering the results discussed in this paper: Clinton Page, Daniel Smit, and Fengyi Yang.

complex patterns from a given data set without the need to first define the important features by hand. This allows for large features sets to be analysed with the possibility of learning unprecedented ways to represent the underlying patterns within the data.

Although deep learning is capable of a generalised form of pattern detection, the performance of any learning technique is still only as good as the data it is trained on. As the implementation of deep learning on network traffic classification is only just starting to emerge, the question of how to best represent network data to such a classifier for effective training still remains. This paper aims to address this question by exploring three different ways of representing data to a deep neural network. The contributions of this paper are as follows:

1. We devise, evaluate and discuss three different representations of network data for use in a deep learning classifier.
2. We highlight the potential for novel deep packet inspection techniques based on deep learning and show that useful predictions of malicious classes can still be made within the first 50 bytes of a packet's payload.
3. We show that deep learning achieves comparable results to other machine learning methods when using only metadata features.

2 Related Work

Network traffic classification techniques typically fall into two categories: payload-based classification, where the traffic is classified based on the distinct signatures found in the message content of the packet; and statistical-based classification, where the traffic is classified based on a collection of metadata features, such as the number of packets sent. The latter of these two options has been typically favoured in machine learning research in recent years due to the ease of generating well defined features sets [2]. While this technique displays desirable results on detecting malicious network traffic, they rely on existing knowledge of what features certain attacks are likely to exhibit and thus are unlikely to lead to any additional insights into the data.

Payload-based classification has been typically left for non-learning algorithms such as deep packet inspection (DPI), which searches the packets for a list of predefined signatures. This method benefits from high classification accuracies but is only as accurate as its knowledge base of such attacks. This static approach remains popular for commercial systems that have the resources to manage such a large repository of known attacks. However, with the introduction of deep learning, payload-based techniques are beginning to make use of learning algorithms that facilitate a more adaptive signature-based detection.

Wang [7] showcased deep learning's potential to achieve an automation of DPI methods, acquiring high classification accuracies for application traffic using the first 1000 bytes of a network flow's payload as the input to a deep (multi-layered) neural network. Likewise, Wang *et al.* [6] used a convolutional neural network (CNN), a deep learning approach typically utilised for image classification, to detect patterns within the payload data. This approach was reported to achieve high classification accuracies for both application and malicious traffic but was unable to be formally validated in comparison to the standard techniques of classification.

In this paper we explore both statistical and payload-based approaches for deep learning in the context of cyber malicious classification and detection. Through our analysis, we show that both techniques have different strengths and weakness, and conclude that a union of the two would result in a more robust classifier.

3 Network Data Representation for Deep Learners

The performance of any machine learning technique is only as good as the input data it receives. Although deep learning is capable of extracting useful features from a larger feature set, an effective representation of the input data will further aid the classifier to perform better on the given task. This section defines three representations of network data for use in a deep learning classifier.

3.1 Payload Data

The main disadvantage of implementing deep packet inspection in a real-world scenario is that the signatures used for classification can be subject to regular change. This is especially evident in malicious attacks as their implementation is constantly evolving in order to bypass existing systems of detection. However, a way of automating this feature selection process is proposed using deep learning. By allowing the deep learning classifier to train from the same data commonly used in DPI, the classifier can attempt to learn the underlying patterns within the payload content rather than just searching for pre-defined signatures. Like DPI, a training process will still need to be re-administered to keep up with the constant evolution of network traffic but deep learning can remove much of the human effort involved in this process.

As the inputs to a deep learning classifier must remain fixed, a selection of the payload data must first be made. In this investigation the first 50 bytes of each traffic flow were mapped to the inputs of the deep learner; each byte value mapping to a corresponding input node. To evaluate this input strategy, a dense neural network was created with two hidden layers of 1000 nodes each. This input method expands upon research outlined in [5].

3.2 Flow Image

The largest defining factor between deep learning and other machine learning techniques is its ability to generate insight from a large source of data. To this effect, the method of data representation described herein aims to maximise the amount of input data seen by the classifier in order for a more complex evaluation of the traffic to be made. However, due to the large size of this input method, a standard dense neural network would not be able to efficiently evaluate this input and therefore another deep learning method must be selected.

Convolutional neural networks (CNNs) have been at the forefront of deep learning due to their performance in image processing tasks. Given that image classification requires the processing of large input sizes, CNNs have developed many ways of increasing their efficiency in these tasks such as utilising dimension reduction layers known as pooling.

In order to explore network traffic with a CNN, each flow from the captured network traffic was first converted to an image with each pixel representing a byte of data in the network. Each row of the image is a new packet in the flow, with the bytes contained in the respective packet filling out the columns.

3.3 Flow Statistics

In recent years, flow statistics has been the conventional subject of analysis when applying machine learning techniques to the field of network traffic classification. This method of classification is based on the assumption that the traffic's metadata can be used to identify the distinguishing behaviours of certain traffic. For example, frequent short messages could be an indicator of a denial of service (DoS) attack. As metadata collected from the traffic is seen as an intrinsic property, this method of analysis was aimed at creating a method of detection that was resilient to obfuscation.

To investigate this method of data representation, 24 features were collected from the traffic. These features consisted of data relating to four main categories, temporal, data transferal, TCP window and IP flags. To analyse these features, a dense neural network was created consisting of two hidden layers of five hundred nodes each. In deep learning, it is common practise to use low level features as inputs to the neural network such that higher order features can be learnt during the training process [1]. However, using these high order features directly was explored to compare deep learning to the more traditional practises.

4 Data Set

Deep learning benefits from a large and extensive data set. For this reason and the need for labelled data, the UNSW-NB15 data set [3, 4] was chosen as the subject of this experiment. Contained in this data set was a set of nine synthetically generated malicious classes. However, as deep learning requires a large number of samples to generate an accurate prediction, four of the smallest malicious classes were merged into a single class entitled OTHER-MALICIOUS. The remaining malicious classes and their distributions within the data set are presented in **Table 1**. A non-malicious class was also included such that the classifier's ability to distinguish between malicious attacks and normal traffic behaviour could be investigated. This non-malicious class was made up of traffic likely to be in abundance on a typical network such as HTTP, P2P, and FTP.

5 Results

In this section, the inputs methods (Section 3.1 to 3.3) are evaluated based on their ability to detect and classify malicious traffic. In order to provide a baseline comparison, three machine learning classifiers, support-vector-machine (SVM), decision tree-based J48, and random forests, were also trained using the Flow Statistics input method.

The F1 scores for malicious traffic classifications are shown in **Table 2**. By examining the weighted-average row in this table, it can be seen that the two highest performing methods are payload-based. As payload-based methods typically rely on security experts to predefine and manage a set of known signatures, deep learning's performance on this input method shows the potential of these techniques to augment or automate this laboursome process. Furthermore, the Payload Data method showcases that a useful prediction can still be achieved with just the first 50 bytes of a packet's payload, allowing for faster predictions to be made.

Using the flow statistics method to compare deep learning to other machine learning techniques, it was shown that given a well-defined task and a strong representative feature set, other forms of tree-based machine learning may outperform certain deep learning techniques. While deep learning can achieve comparable results on this same feature set, due to the complexity in how it correlates input data, we suspect overly deep neural networks may unintentionally obscure some of the underlying features critical for distinguishing malicious traffic.

Although low accuracies are shown for DoS attacks for all methods, its low detection rate in classifiers which showed an overall high performance (i.e. Payload Data and Flow Image) highlights the issue of choosing one technique over the other. As a DoS attack will typically flood the network with benign packets, a detection system which only analyses payload data will not be able to recognise such an attack.

6 Conclusion and Future Work

In this paper three different ways of representing network data to a deep learning classifier were explored in the context of malicious traffic classification. As

Table 1. UNSW-NB15 malicious classes.

Malicious Classes	Number of Samples	Percentage of Total
DoS	3,603	3.1%
Exploits	25,274	21.8%
Fuzzers	19,240	16.6%
Generic	3,798	3.3%
Reconnaissance	11,671	10.1%
Other-Malicious	2,249	1.9%
Non-Malicious	50,000	43.2%

Table 2. Malicious classification comparison with F1 scores.

Malicious Classes	Deep Learning Methods			Other Machine Learning Methods		
	Payload Data	Flow Image	Flow Statistics	SVM	J48	Random Forest
DoS	36.4	36.5	39.3	0.5	**52.2**	**52.2**
Exploits	88.8	**92.2**	85.3	72.2	87.6	88.0
Fuzzers	94.6	**95.1**	86.3	61.3	90.9	89.7
Generic	**88.7**	80.6	69.6	0.0	83.9	87.2
Reconnaissance	96.9	**98.2**	83.6	47.0	84.5	83.7
Other-Malicious	86.6	**92.8**	37.4	0.0	54.0	54.1
Non-Malicious	97.7	**99.1**	98.9	98.9	98.9	98.9
Weighted-Average	92.7	**94.2**	88.3	73.4	90.8	90.8

to be expected, there is no one size fits all solution, with different malicious attacks exhibiting different defining characterises. While deep learning does not show a significant improvement to other, more conventional machine learning approaches for statistical-based malicious traffic detections, its introduction has paved new grounds for the automation of payload-based detection. In future works, the combination of statistical and payload-based inputs will be explored.

References

1. Bromley, J., Guyon, I., LeCun, Y., Sckinger, E., Shah, R.: Signature verification using a “siamese” time delay neural network. In: Advances in Neural Information Processing Systems. pp. 737–744
2. Divyatmika, Sreekesh, M.: A two-tier network based intrusion detection system architecture using machine learning approach. In: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT). pp. 42–47. https://doi.org/10.1109/ICEEOT.2016.7755404
3. Nour, M., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). Military Communications and Information Systems Conference (MilCIS) (2015)
4. Nour, M., Slay, J.: The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. Information Security Journal: A Global Perspective pp. 1–14 (2016)
5. Smit, D., Millar, K., Page, C., Cheng, A., Chew, H.G., Lim, C.C.: Looking deeper using deep learning to identify internet communications traffic. In: 2017 Australasian Conference of Undergraduate Research (ACUR)
6. Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning. In: 2017 International Conference on Information Networking (ICOIN). pp. 712–717. https://doi.org/10.1109/ICOIN.2017.7899588
7. Wang, Z.: The applications of deep learning on traffic identification (2015), https://goo.gl/WouIM6

Statement of Authorship

Title of Paper	Using Convolutional Neural Networks for Classifying Malicious Network Traffic
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Using Convolutional Neural Networks for Classifying Malicious Network Traffic," in Deep Learning Applications for Cyber Security: Springer, 2019, pp. 103-126.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	~~9/8~~ 8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Using Convolutional Neural Networks for Classifying Malicious Network Traffic*

Kyle Millar[1], Adriel Cheng[1, 2], Hong Gunn Chew[1] and Cheng-Chew Lim[1]

1. School of Electrical and Electronic Engineering, The University of Adelaide, Australia
2. Cyber and Electronic Warfare Division, Defence Science & Technology Group, Australia

{kyle.millar, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au
adriel.cheng@dst.defence.gov.au

Abstract. As the reliance on the Internet and its constituent applications increase, so too does the value in exploiting these networking systems. Methods to detect and mitigate these threats can no longer rely on singular facets of information, they must be able to adapt to new threats by learning from a diverse range of information. For its ability to learn complex inferences from large data sources, deep learning has become one of the most publicised techniques of machine learning in recent years. This chapter aims to investigate convolutional neural networks (CNNs), a deep learning technique typically reserved for image classification, and how its methodology can be adapted to detect and classify malicious network traffic.

Keywords: Convolutional neural networks, deep learning with GPUs, malware classification and detection, analysis and similarity.

1 Introduction

The interest in deep learning over traditional machine learning algorithms originates from its ability to generate complex representations of the input data without the need to hand-select a feature set. This process of making complex decisions directly from the source of information has labelled deep learning an 'end-to-end' algorithm [3]. Often, conventional learning techniques may inadvertently discard useful knowledge if subjected to a feature selection process based on the heuristics of a human operator. However, deep learning aims to draw new inferences about a data source, in our case Internet Protocol (IP) network traffic, that might not have been used otherwise.

This new wave of deep learning research has been largely popularised by the field of image classification. Deep learning's ubiquity in image classification was largely

* This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence. The experiments described herein were enabled by the supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

sparked in 2012 when a deep convolutional neural network (CNN) won the ImageNet visual recognition challenge, almost halving the current error rate at the time [4]. Since then, almost all subsequent submissions to the ImageNet challenge have been made using a variation of these CNN learning techniques [5].

Convolutional neural networks are a variant of artificial neural networks which were designed to leverage insights from locally connected features within the input. Although largely used in the field of image classification, CNNs have been successfully applied to other application areas such as natural language processing [6-9], audio classification [10], and signal processing [11].

Convolutional neural networks have recently gained interest in Cybersecurity and network management research. They have shown promising results in this field, able to perform both statistical-analysis, by interpreting traffic metadata features [12, 13], and facilitate payload-analysis by finding patterns within the message content of the traffic [3, 14, 15]. Despite the potential of CNNs, there has yet to be an in-depth study on the principles of designing and implementing a CNN for use in network traffic classification.

The aim of this chapter is to introduce the theory behind CNNs with an emphasis on why its distinct properties make it well suited to detect evasive malicious network traffic. In particular, the contributions of this chapter are as follows:

- We outline three benefits of utilising CNNs and examine how these properties could both benefit or hinder network traffic classifications in a Cybersecurity context.
- We introduce a novel way of representing network traffic to a deep learning classifier inspired by natural language processing - entitled a Flow-Image.
- We compare three different CNN architectures including a novel architecture aimed at exploiting the known properties of a TCP/IP packet.
- We explore the structure of network traffic and propose how CNNs could be employed to create a classifier that is robust to certain malicious evasion techniques.

This chapter assumes the reader has prior knowledge and background in neural networks and their terminology; otherwise, we refer the reader to [16] for an in-depth introduction to this topic.

The remainder of this chapter is as follows. Section 2 introduces the key concepts and advantages behind a CNN implementation. The advantages of a CNN based implementation are then explored in terms of their application to malicious network traffic classification in Section 3. Section 4 outlines how network traffic has been represented to CNNs in related literature, providing an additional novel representation used to address the issues found with existing methods. Section 5 examines three CNN architectures and details how they can be used for network traffic classification. Section 6 provides a brief summary of the experimental setup used for analysis in this chapter. Section 7 presents experimental results and describes how different properties of CNNs can affect the classification of malicious network traffic. The chapter concludes in Sections 8 and 9 including suggestions for future research.

2 Convolutional Neural Networks

The underlying assumption of a CNN is that information within the input is locally correlated [17]. For example, when reading a sentence, it is not just the individual letters that are relevant but the relationship between them. This relationship is often diminished by distance, with the correlation between letters in one word having little to no effect on the words that follow. Finding local correlations is often much more efficient than analysing the entire input at once. In artificial neural networks, these local correlations can be enforced by using local receptive fields. **Local receptive fields** restrict the inputs to a node to a small localised region of the overall input. Figure 1 depicts how a local receptive field can be used to restrict the input of a node to a subsection of the previous layer.

There are several advantages to utilising local receptive fields. Firstly, the entire input does not have to be evaluated at once. This is particularly important when working with large input sizes where, in classical deep learning techniques such as a fully-connected neural network, the size of the neural network scales significantly with the number of input fields used. Larger neural networks result in larger hardware requirements and can cause overfitting if there are too few samples to train on [18].

The second advantage occurs when there is translation or repetition of important features within the input. An example of this advantage is prevalent in object detection tasks, where an object should be just as easily detected no matter where in the image it occurs. Once a local correlation is identified, local receptive fields act as a search function, finding that same correlation regardless of its location in the input; therefore, if a local correlation is either shifted or repeated, no additional complexity needs to be added to the neural network to identify it. This is known in artificial neural networks as **weight-sharing**. Depending on the application, weight-sharing can vastly reduce the

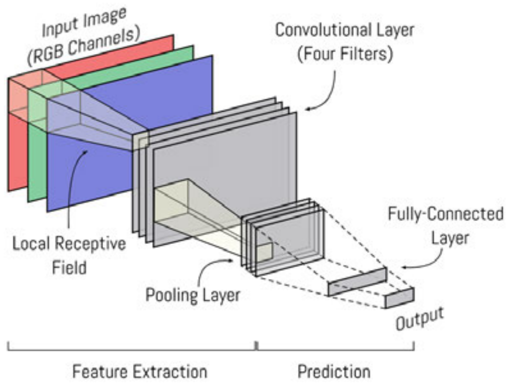


Figure 1. The structure of a typical CNN used for image classification. The convolutional and pooling layers form the feature extraction stage while the fully-connected layers form the prediction stage.

size of the neural network and can help increase its ability to generalise to new data [18].

The third advantage of local receptive fields is that they are more robust to localised noise. In many classification tasks, the input is likely to suffer from a factor of noise. For example, voice recording can often suffer from ambient high or low frequency interference. A system that evaluates solely based on the overall input is likely brittle to these occurrences; however, it is more likely that only certain sections of the input is affected. Utilising local receptive fields can help localise the effect of noise to the regions in which they occur [7].

Convolutional neural networks are simply neural networks that enforce the use of local receptive fields. This is achieved using two specialised layers, the **convolutional layer** and the **pooling layer**. A typical CNN is made up of two stages. The first stage utilises convolutional and pooling layers to identify local correlation within the input and allow for weight-sharing. This stage can be thought of as learning and extracting features from the input. The second stage utilises fully-connected layers to combine the features found in the previous stage into an overall prediction. This stage is thus thought of as the predication stage. Figure 1 shows a graphical representation of a typical CNN structure used for image classification, highlighting the feature extraction and prediction stages.

In the remainder of this section, each of these layers are briefly introduced as prerequisites for later sections of the chapter. However, for a more in-depth explanation of CNNs and their structure, the reader is referred to [17, 19].

2.1 Convolutional Layer

The convolutional layer is the core building block of a CNN. A convolutional layer enforces the idea of local receptive fields by limiting the scope of each node in its layer to a select few nodes in the preceding layer.

A feature of local receptive fields stated in the previous section was that of weight-sharing, the ability to detect repeated or translated patterns without adding additional complexity into the network. A convolutional layer implements this ideal using a **filter**, a sliding function used to detect the same pattern in different regions of the input. In Figure 2, a single filter is depicted as it scans across the input for a particular pattern. To search the input for a varying number of patterns, a convolutional layer will typically comprise of many filters, the total output of which is entitled a feature map**.** The **feature map** can be thought of as identifying the location in which each filter found their respective pattern. An example feature map consisting of four filters is depicted in the convolution layer in Figure 1.

The dimensionality of a CNN refers to the dimensions of the filters in its convolutional layers. This is often related to the dimensionality of its input. One-dimensional (1D) and two-dimensional (2D) CNNs are typically seen in research due the magnitude of tasks in which one or two dimensional arrays are used for inputs. 2D-CNNs are typically used in image classification where the two dimensions relate to the spatial coordinates of an image, whereas 1D-CNNs are generally used for sequential inputs such as text classification. Figure 1 and Figure 2 provide an example of a 2D-CNN and a 1D-CNN respectively. It should be noted that the depth of the filter is not considered as a dimension in this terminology. An example of filter depth is given in Figure 1 where a filter depth of three is used to analyse the three input channels of the image (red, green, and blue).

2.2 Pooling Layer

The pooling layer serves two purposes, (i) it reduces the size of the feature map and (ii) provides a level of translation invariance to the features found therein. The most important values of the feature map are where the filters detect their respective patterns. This is identified by strong positive or negative values within the feature map. The larger the absolute number is, the more likely that the filter has detected a pattern in that location. The pooling layer reduces the percentage of low values in the feature map by summarising subregions by their important values.

Max pooling is the most commonly utilised pooling function. It samples subregions in a similar process to that of the convolutional layer; however, instead of finding patterns in the input, it takes the maximum absolute value within that subregion. This increases the density of important values in the feature map whilst also allowing for slight translation in the locations of the patterns found. Figure 1 illustrates a pooling layer sampling the previous layer to reduce the size of its feature map.

2.3 Fully-Connected Layer

The fully-connected layers used in a CNN are the same as found in a typical fully-connected neural network. Each node in this layer is connected to every node in the previous layer. This structure allows for the extraction of global relationships from the

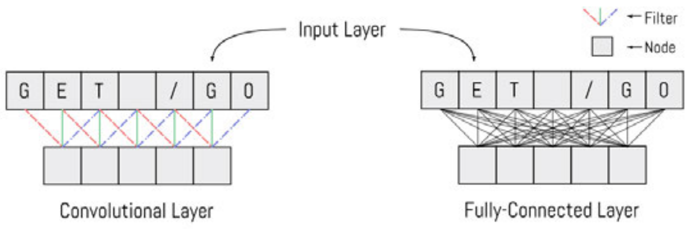


Figure 2. A comparison of the connections in a convolutional layer versus a fully-connected layer. Each node in the convolutional layer makes the same three connections to the nodes in the input layer, albeit at different locations. This is known as a convolutional layer with a one-dimensional filter size of three. In contrast, every node in the fully-connected layer is connected to every node in the input layer.

previous layer. A CNN uses fully-connected layers to combine the individual features found in the convolutional layers into a final prediction of the overall input.

The disadvantage of this layer is that the number of connections is proportional to the number nodes in the previous layer. This can cause significant computational overhead when input sizes are large; therefore, convolutional and pooling layers are often designed to reduce the number of parameters in the input before a fully-connected layer is used.

3 Benefits of CNNs for Network Traffic Classification

In this section, we identify three benefits of CNNs for the classification of network traffic in a Cybersecurity context:

- Efficiency on larger input sizes (Section 3.1).
- Translation and repetition invariance (Section 3.2).
- Robustness to noise (Section 3.3).

3.1 Efficiency on Larger Input Sizes

Network traffic is typically characterised by large volumes of traffic. Packets spanning the maximum transmission unit (MTU) of 1500 bytes and network traffic flows using hundreds or thousands of packets are commonly seen in today's networks. It is therefore necessary to consider how to efficiently evaluate such large input sizes when using neural networks.

The **trainable parameters** of a neural network are the parameters that are tuned while training to reduce its error on the desired objective [16]. The most important trainable parameter is the weight of the connections between nodes. Every connection in the neural network has an associated weight. The value of this weight is adjusted during training to determine the significance of the connection to the overall classification. A CNN can considerably reduce the number of trainable parameters of a neural network by using weight-sharing. Weight-sharing allows a trained weight value to be shared between multiple connections in the same layer.

In the previous section, the relationship between the input to a fully-connected layer and the number of connections in the neural network was highlighted. The computational requirements of maintaining such a large number of trainable parameters for each of these connections would rule out many hardware implementations and could lead to overfitting if there are too few samples to train on [18].

Figure 2 highlights the comparison between the number of trainable parameters in a convolutional layer versus a fully-connected layer. Although there are 15 connections made in the convolutional layer, due to weight-sharing, only three weights require adjusting. In contrast, a fully-connected layer with the same number of nodes would result in 35 adjustable weights.

The reduction in trainable weight parameters as a result of weight-sharing allows a CNN to be much more computationally efficient than other types of neural networks when dealing with larger input sizes. This result is further analysed in Section 7.1.

3.2 Translated and Repeated Patterns

The process of scanning for discrete, local patterns enables a CNN to locate features regardless of where they appear in the input. The benefit of this approach has been shown in speech recognition tasks where the varying speed, tone, and inflection of a speaker can change the positioning of these identifying features [18]. Malicious network traffic has been shown to exploit similar properties in the size, sequencing, and timing of packets to obfuscate the identifying features of malicious intent [20].

In allowing for variations within the input, a CNN can be robust to certain evasion techniques. Table 1 shows a simple example of how different string manipulations could be used to extract the same information. In order to learn to detect this command, a fully-connected neural network would have to learn each of these representations individually. In contrast, a CNN could reuse the same patterns it has found in its lower levels of abstraction to quickly generalise to new variants.

Weight-sharing allows a CNN to identify repetition in the input without adding any additional complexity to the neural network. From a network traffic perspective, similar syntax structures and duplicated packets make repeated patterns in data and communication traffic highly prevalent.

3.3 Noisy Inputs

In many classification tasks the input is likely to suffer from a factor of noise; however, what is noise in terms of malicious network traffic is hard to define. On one hand, randomly generated strings could be injected into a packet to confuse the classifier's decision process; but, on the other hand, if these randomly generated strings are known to be classified as noise, they could be used to hide malicious content.

The use of local receptive fields allows a CNN to classify an input even when certain sections of the input have been obfuscated by noise [7]. This property would allow a CNN to be robust to certain injection attacks where noise is added to confuse a detection system; however, this property could also be exploited.

The convolutional layers of a CNN scan the input for identifying features. If the features of malicious intent are not known, it is unlikely that any warnings would be

Table 1. Examples of simple string manipulation [1].

Method	**Example**
Self-Referencing	`/_vti_pvt/././././administrators.pwd`
Double Slashes	`/_vti_pvt///administrators.pwd`
Reverse Traversal	`/scripts/../_vti_pvt/administrators.pwd`

raised; however, a detection system which accounts for the entire input might see a change in the overall input and raise suspicion. A CNN's ability to be robust to noise could therefore both benefit and hinder the classification of malicious network traffic, depending on the evasion techniques utilised.

4 CNN Input Representations

In this section, we investigate how network traffic can be represented to a CNN. In particular, this section evaluates the current methods of representation in related literature and introduces a novel approach that aims to improve the efficacy of using a CNN for network traffic classification.

4.1 Related Work

Network traffic classification typically falls into two categories; (i) statistical-based classification, where the traffic is classified based on a collection of metadata features, such as the number of packets sent; and (ii) payload-based classification, where the traffic is classified based on the distinct signatures found in the message content of the packet [15].

Statistical-based approaches were explored using a CNN in [13]; however, no prior consideration was undertaken as to what the best representation should be to fully exploit the strengths of a CNN. In this chapter, we address this directly and strive to utilise the full potential of a CNN.

Wang *et al.* [3, 14] explored the use of a CNN for payload-based classification in which a 28x28 pixel image was used to represent the first 784 bytes of the payload content of a traffic flow. Each byte was represented by a greyscale pixel using the range from black (represented by the value 0) to white (represented by the value 255). This provides a one-to-one mapping with the 256-value range of a byte.

Wang *et al.* showed that a 1D-CNN achieved better classification accuracy than a 2D-CNN on this input representation. Although not stated by the authors, this result is likely to have stemmed from how the second dimension of the image was created. The payload was wrapped to form the second dimension of the image; therefore, this dimension was simply an extension of the first. Analysing a one-dimensional input with a 2D-CNN is likely to reduce the classifier's ability to generalise to new data and may have caused the reduction in accuracy reported.

The value of a pixel in an image is numerical. In greyscale, this value represents a scale between black and white. Representing a byte as a greyscale pixel infers the same scale between its values. For example, implying that a byte value of zero has more similarity to a value of one than it does to ten. While this representation was shown to hold for a test data set in [3, 14], it is not always a true property of the values in a byte for different types of network traffic. Obfuscation techniques of malicious traffic aim to exploit the difference in how a detection system and its intended target process data. Using these differences, a malicious attack can hide its true intention from the detection system while still delivering its malicious payload to its target. A numerical representation of bytes in a detection system could be easily exploited as it is using a property that is likely to be non-existent in the intended application.

To address the above issues, a novel approach to network data representation was explored as part of this research. This input representation will be referred to as Flow-Image for the remainder of the paper.

4.2 Flow-Image

Network traffic is fundamentally a conversation between two endpoints; therefore, rather than taking inspiration from image classification as per the related work above, we seek inspiration from natural language processing (NLP).

Text classification is one of the cardinal topics for NLP in which text is classified based on a predefined objective such as sentiment analysis (i.e., identifying whether the writer's opinion is positive or negative) or terminology extraction (i.e., identifying important words in a larger body of text). These methods typically make use of feature generation techniques, such as bag-of-words, to represent language in higher order representations before analysis [9]. This however requires predefined knowledge of the syntax of the language you are classifying.

To classify malicious network traffic, assumptions based on syntactic and semantic structures cannot be made as malicious attacks can exploit these assumptions to circumvent known techniques. Fortunately, CNNs have recently been investigated to perform classification without any predefined knowledge of the subject language [8, 9]. This is achieved by analysing the text at its fundamental level, i.e. its individual characters.

Character-level NLP allows the classifier to learn the complete formation of the text directly from its underlying characters. While this adds additional complexities to the network, it allows for the classification of text when the structure of the language is unknown. Zhang *et al.* [9] showed that CNNs can be successfully applied to character-level NLP tasks, achieving comparable results to the state-of-the-art techniques for English text classification.

To represent characters to a neural network, a categorical representation must be used as there is no intrinsic correlations between the values of a character. **One-hot-encoding** is a common way to represent categorical information as it is a simple way to represent non-ordinal categorical data. One-hot-encoding constructs a vector of size *n*, where *n* is the number of distinct categories an input value can take. Each entry in this vector is zero except for the entry that corresponds to the category that this vector represents, which is given a value of one. For example, in network traffic, to represent the byte value '0', a vector of size 256 would be constructed with the first entry set to one and the rest zero. One-hot-encoding was shown in [9] to effectively represent 70 different characters to a CNN. This method will be used in Flow-Image to represent the 256 distinct values of a byte.

Current implementations of CNNs require a fixed input size. Therefore, the number of bytes analysed in each flow must be set before analysis. Furthermore, to investigate the effect of a 2D-CNN on network traffic classification, the input must also have two representative dimensions. To address these two factors, a network traffic flow was represented in a two-dimensional array as shown in Figure 3. Each row of this array represented a new packet in the flow with each column representing a new byte in the packet. The array was fixed such that 97 bytes of the first ten packets of a bidirectional flow were analysed. Flows that did not meet the required array size were padded with byte values of zero.

Composing the 97 bytes of a packet were 47 bytes from the TCP/IP header and 50 bytes from the payload. The header section was included in the Flow-Image as some attacks are easier to detect by their header sections (i.e., scans and probes), while others are easier to detect by their payload (i.e., worms) [21].

The number of bytes in the TCP/IP header section was limited to 47 as this was found to be the maximum header length for over 99% of the packets in the data set used for analysis (further details of the data set are given in Section 6.1). The 47-byte header encompassed an 11-byte IP header (source and destination IP addresses, and TTL fields were removed due to synthetic biases found in the data set) and a 36-byte TCP header.

The payload was truncated to the first 50 bytes of each packet as it has been shown that the classification significance of a byte in the payload decreases the further into the payload it is found [22-24]. The number of flows in the Flow-Image was truncated to ten to balance classification efficiency with the ability to analyse the relationship between packets in a flow.

It should be noted that the assumptions on the size and number of packets to evaluate were based on analysis of the data set used. These assumptions may not hold under other circumstances. For example, a malicious attack could easily circumvent detection by sending malicious content outside the limits of the Flow-Image stipulated in this chapter. Before a detection system is deployed, a decision should be made between the efficiency of the system and its robustness to attacks looking to exploit this property.

To represent bytes in a one-hot-encoded vector, a third dimension to this array was added. Figure 3 shows a graphical representation of this Flow-Image in contrast to how a typical image would be represented to a neural network.

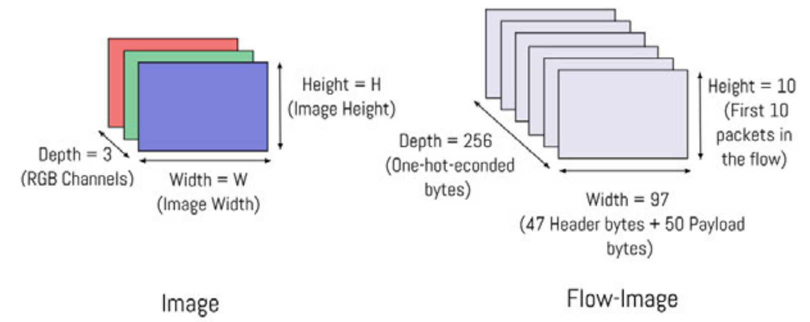


Figure 3. Representation of an input used for image classification compared to Flow-Image.

5 CNN Architectures

In this section, we explore three CNN architectures and examine how the properties of CNNs can be exploited to improve the detection and classification of evasive malicious network traffic. The configuration selected for each CNN architecture is shown in Table 2.

5.1 2D-CNN

A 2D-CNN is the most common architecture due the abundance of CNNs used for image classification. Images consist of two dimensions which both relate to the same spatial coordinate system; therefore, local correlations that make use of both dimensions must exist. A 2D-CNN is able to exploit this property by scanning for multi-dimensional correlations.

The Flow-Image input has two representative dimensions. The first represents packets in the flow and the second represents bytes in a packet. A 2D-CNN could be utilised to find locally correlated bytes both within the same packet and in adjacent packets. Network traffic typically emulates a call and response within its

Table 2. Structure of the three CNN architectures. Each convolutional layer consists of 256 filters. Max pooling is used for all pooling layers.

- Conv[*h*×*w*] – Convolutional layer with a filter height and width of *h* and *w* respectively.
- Pool[*h*×*w*] – Pooling layer with a filter height and width of *h* and *w* respectively.
- FC[*n*] – Fully-connected layer with *n* nodes. Seven nodes were used in the output layer to correspond with the seven classes of the evaluated data set (Section 6.1).

Layers	2D	1D	Segmented-CNN	
			Header	Payload
1	Conv[3×3]	Conv[1×3]	Conv[1×47]	Conv[1×3]
2	Conv[3×3]	Conv[1×3]	Conv[1×1]	Conv[1×3]
3	Pool[2×2]	Pool[1×2]	Pool[2×1]	Pool[1×2]
4–5	Conv[3×3]	Conv[1×3]	Conv[1×1]	Conv[1×3]
6	Pool[2×2]	Pool[1×2]	Pool[2×1]	Pool[1×2]
7	FC[256]	FC[256]	FC[256]	FC[256]
8	FC[1024]	FC[1024]	–	FC[1024]
Output	FC[7]	FC[7]	FC[7]	

communication. Most packets are followed by a response or acknowledgement from the other endpoint. Analysing locally correlated packets may yield information into how the two ends of the connection are communicating. Figure 4 depicts the key properties of the 2D-CNN architecture as it evaluates the Flow-Image input.

5.2 1D-CNN

1D-CNNs have been typically used in NLP tasks where information can be represented as a sequential one-dimensional array. Although there are two representative dimensions in the Flow-Image input, the bytes between a packet have a higher degree of correlation than the bytes over multiple packets. A 1D-CNN could therefore be used to focus on the stronger correlation found between bytes in a packet. A diagram of the 1D-CNN architecture is shown in Figure 4.

5.3 Segmented-CNN

A Segmented-CNN is a novel CNN architecture that aims to capitalise on the distinct properties of the two sections in a TCP/IP packet - the header and the payload. This architecture is inspired by the approach taken by Bromley *et al.* in their design of a Siamese-CNN [25]. A Siamese-CNN utilises two CNNs to simultaneously analyse two distinct images, the output of which is then compared to determine the similarity between the two. However, instead of measuring the similarity between these two

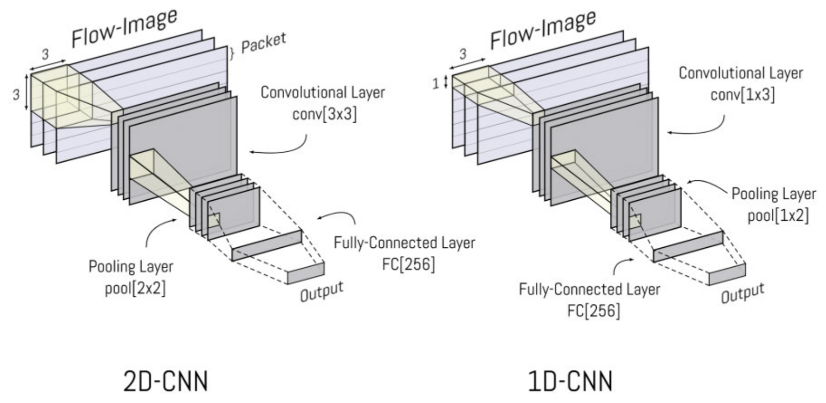


Figure 4. Structure of a 2D-CNN and 1D-CNN. The filters of a 2D-CNN span multiple packets, while a 1D-CNN's filters only consider the bytes of a single packet. Only the first convolutional, pooling, and fully-connected layers in the 2D-CNN and 1D-CNN have been shown.

images, this methodology could be adapted to analyse two inputs that relate to the same classification but have their own distinct properties. Figure 5 illustrates how two CNNs could be used to analyse the distinct sections of a TCP/IP connection.

The header section of a packet relates to the information required for it to traverse a network. It has a strict structure that needs to be enforced such that packets can be received intact at the other end of the connection. Therefore, there is very little variance in how the header section can be constructed. Furthermore, as most values of this input relate to a distinct property of the packet, there is almost no local correlation within the header section. The classification of the header section of a packet is therefore more suited to a fully-connected neural network. However, multiple headers in a flow are likely to exhibit similar properties and therefore benefit from the utilisation of weight-sharing. To optimise a CNN for these two properties, a one-dimensional filter which spans the entire header section of a packet was used. At the packet level, this filter acts as a fully-connected neural network; however, the use of a one-dimensional filter allows for weight-sharing between the packets.

The payload is the message content of a TCP/IP packet; therefore, a CNN structure inspired by natural language processing should be used. The 1D-CNN (Section 5.2) was selected to be used for the payload section of the Segmented-CNN.

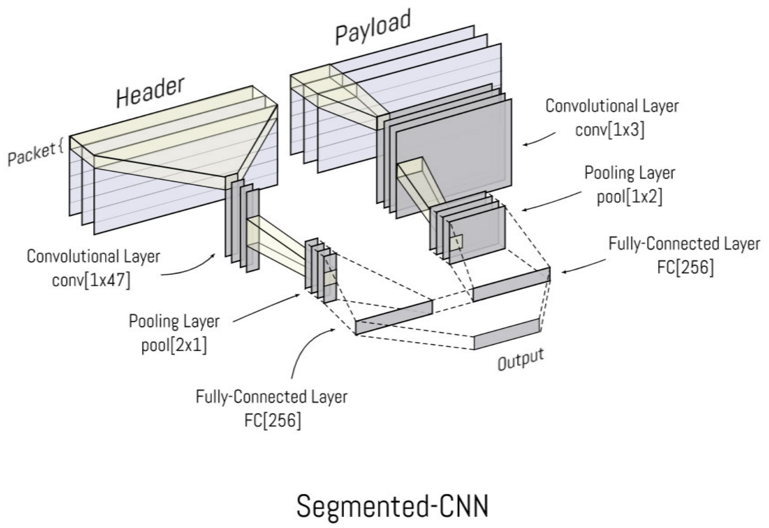


Figure 5. Structure of a Segmented-CNN illustrating how two CNN networks can be individually optimised for the two distinct regions of a packet - the header and payload. Only the first convolutional, pooling, and fully-connected layers of the Segmented-CNN have been shown.

6 Experimental Setup

6.1 Data Set

To evaluate CNNs on network traffic, a data set that contained full packet captures was required. For this reason the UNSW-NB15 data set [26, 27] was chosen as the subject of our experiments. Contained in this data set was a set of nine malicious classes. However, the four smallest malicious classes contained too few samples to reliably train our techniques on. Instead of removing these malicious classes from our analysis, they were merged into a single class entitled OTHER-MALICIOUS.

A description of each malicious class as provided by the UNSW-NB15 data set has been presented in Table 3. It is seen that the provided descriptions introduce a level of confusion into this evaluation. For example, the ANALYSIS malicious class is described as an attack that utilises port scans to penetrate web applications; however, port scans are more commonly associated as part of a RECONNAISSANCE style attack. Additionally, GENERIC is not a formalised malicious attack variant. It appears that this class has been defined by UNSW-NB15 as a collection of various collision-based attacks.

The descriptions of malicious classes as given by UNSW-NB15 highlights the complexity of multiclass classification of malicious network traffic. The constant evolution of malicious attacks has increased the difficulty in defining them. Moreover, sophisticated attacks will often incorporate different elements from across multiple malicious classes; thereby making taxonomy ineffective.

The analysis of different malicious classes was not the subject of this experiment; therefore, the malicious classes as defined by UNSW-NB15 were not altered. However, a non-malicious class was introduced such that the classifier's ability to distinguish between malicious attacks and normal traffic behaviour could be investigated. This non-malicious class was composed of benign traffic collected from the UNSW-NB15 data set.

The UNSW-NB15 data set was split into three subsets, training (72%), validation (8%), and testing (20%). The distribution of each class in our analysis has been provided in Table 4.

6.2 Hardware and Software

The recent rise of deep learning research has been largely spurred by advancements in the specialised hardware required to utilise it. Previously, the sheer number of mathematical operations within a deep learning algorithm reduced its applicability on computationally or temporally limited applications; however, with the introduction of dedicated graphics processing units (GPUs), many of these operations can now be processed in parallel.

Table 3. Class descriptions as given in the UNSW-NB15 data set [2].
It should be noted that these descriptions have been altered to improve interpretability.

Malicious Classes		**Description**
DOS		An intrusion which disrupts computer resources in order to prevent authorised requests from accessing a device.
EXPLOITS		A sequence of instructions that take advantage of a glitch, bug or vulnerability, causing an unintentional or unsuspected behaviour on a host or a network.
FUZZERS		An attack in which the attacker attempts to discover security loopholes in an application, operating system or a network by feeding it with random data.
GENERIC		A technique that can be established against every block-cipher using a hash function to cause a collision without knowing the configuration of the block-cipher.
RECONNAISSANCE		An attack which gathers information about a computer network to evade its security controls.
OTHER-MALICIOUS	ANALYSIS	A type of intrusion that penetrates web applications via ports (e.g. port scans), emails (e.g. spam) and web scripts (e.g. HTML files).
	BACKDOOR	A technique to gain unauthorised access to a device by bypassing normal authentication procedures.
	SHELLCODE	Malware in which the attacker penetrates a slight piece of code starting from a shell to control the compromised machine.
	WORM	An attack in which the attacker replicates itself to spread on other devices.

Table 4. Distribution of classes from the UNSW-NB15 data set.

Malicious Classes	**Training (72%)**	**Validation (8%)**	**Testing (20%)**	**Total**	
DOS	2,592	290	721	3,603	(3.1%)
EXPLOITS	18,189	2,030	5,055	25,274	(21.8%)
FUZZERS	13,851	1,541	3,848	19,240	(16.6%)
GENERIC	2,727	311	760	3,798	(3.3%)
RECONNAISSANCE	8,397	939	2,335	11,671	(10.1%)
OTHER-MALICIOUS	1,611	188	450	2,249	(1.9%)
NON-MALICIOUS	36,000	4,000	10,000	50,000	(43.2%)
Total	83,367	9,299	23,169	115,835	(100.0%)

The University of Adelaide's Phoenix High Performance Computing (HPC) service was used to train the neural networks evaluated in this chapter. Each neural network was trained on a NVIDIA Tesla K80 GPU, with 32GB of memory and an 8-core central processing unit (CPU). The neural networks were implemented using Google's deep learning framework, TensorFlow [28].

7 Experimental Results

To evaluate the effectiveness of Flow-Image and our CNN architectures outlined in Sections 5.1 to 5.3, we conducted malicious network traffic classification experiments using the UNSW-NB15 data set. A fully-connected neural network (FC) was also analysed to compare the devised CNN techniques to a shallow neural network. The fully-connected neural network consisted of three fully-connected layers of 256, 1024, and 7 nodes respectively.

7.1 Architecture Efficiency

One of the key benefits of CNNs is their efficiency on large input sizes. To identify the extent this has on the classification of the Flow-Image input, the training time and neural network size of each architecture were examined.

Figure 6 shows the number of trainable parameters in each neural network to the nearest million. The number of trainable parameters is roughly proportional to the size of the network in memory (shown in Figure 7). The CNN architectures show between three and ten times reduction in the memory requirements of a neural network. A CNN is therefore better suited for any hardware limited implementations

Figure 8 depicts the training time of each neural network. The Segmented-CNN completed training in the shortest amount of time, less than half the training time of the 1D-CNN. The extended training time of the 1D-CNN suggests that the correlation between packets is harder to detect in later stages of the neural network.

The Segmented-CNN completed training in the shortest time. This result may be specific to the Flow-Image input representation. The pre-processing of network traffic to form the Flow-Image input representation allowed for the separation of the header and payload of each packet into distinct sections of the input. The design of the Segmented-CNN therefore may have expediated the training process by continuing this separation of the header and payload sections within the neural network itself. This allowed for two smaller inputs that had a higher degree of similarity to be analysed. This process is akin to that of a divide and conquer approach.

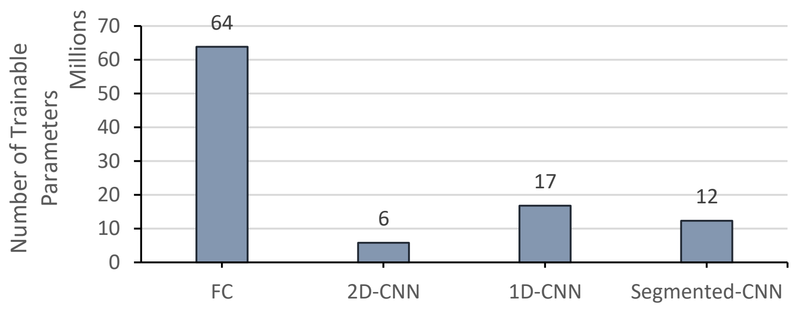


Figure 6. Number of trainable parameters in each neural network, to the nearest million.

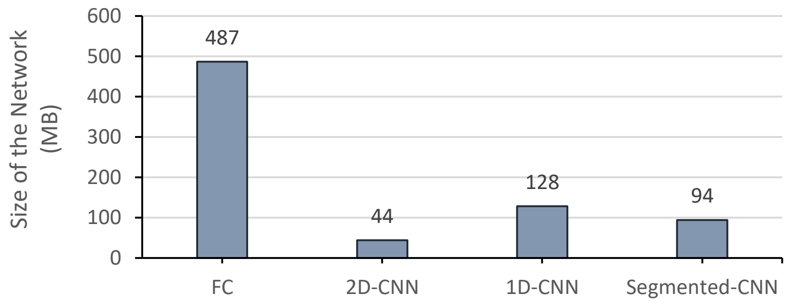


Figure 7. Approximate size of the neural network in memory.

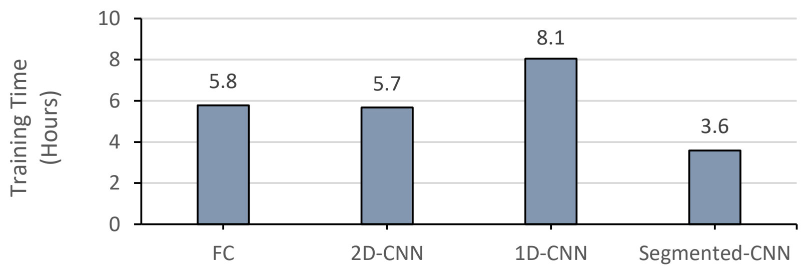


Figure 8. Training time of each neural network.

7.2 Classification Performance

The trained classifiers were evaluated on the testing set, the results of which are shown in Figure 9. All classifiers exhibited similar classification performances on the testing set with the fully-connected neural network achieving the highest classification accuracy overall. The performance of the fully-connected neural network on the testing set suggests that the size of the neural network is sufficient to account for any variation in network traffic within the UNSW-NB15 data set.

The fully-connected neural network achieved almost 100% training classification accuracy on the training set. This result signifies that the classifier was overfitting the training data, supporting our analysis in Section 3.1 that even a small fully-connected neural network would likely overfit the Flow-Image input due to the large number of trainable parameters that would be created.

All three CNN architectures shown in Figure 9 are able to achieve similar test classification performance to the fully-connected neural network albeit with three to ten times fewer trainable parameters. This result suggests that the use of CNNs for network traffic classification would still be recommended over that of a fully-connected neural network when resources are limited.

To provide further analysis, the precision and recall of each class was calculated. **Precision** is the number of correct classifications of a class divided by the number of times that class was predicted. **Recall** is the number of correct classifications of a class divided by the total number of samples of that class. The harmonic average of precision and recall is the F_1-score and is commonly used to compare the performance of a classifier on the individual classes present.

$$\text{F}_1\text{-score} = 2\,\frac{(Precision \times Recall)}{(Precision + Recall)} \tag{1}$$

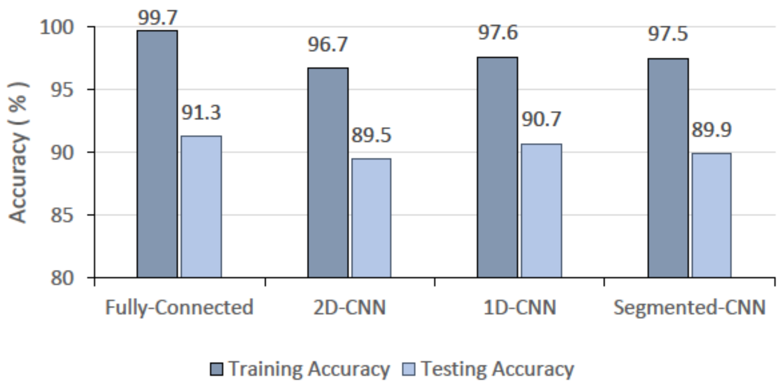


Figure 9. Training and testing accuracy of different CNN architectures on the UNSW-NB15 data set. A fully-connected neural network was also analysed to provide a baseline comparison to the CNN techniques.

Shown in Figure 10 is the F_1-score of the individual classes for each classifier on the testing set. From Figure 10, a pattern in the DOS, GENERIC, and OTHER_MALICIOUS classes was observed. The fully-connected neural network and 1D-CNN achieve the highest accuracies in these classes, followed by the Segmented-CNN and 2D-CNN respectively. This result correlated with the degree of pooling used in each architecture therefore suggesting that the network traffic flows in the UNSW-NB15 were highly structured.

7.3 Network Traffic Structure

To further investigate how the structure of network traffic affects the classification performance of a neural network, the prediction process of the trained classifiers was evaluated.

To train a neural network, the weights of its connections are adjusted depending on their significance to the overall classification. An estimation of the importance of an input to a neural network can therefore be identified by the summation of the weight of its connections.

The process of identifying significant bytes in a TCP/IP flow by the weight of its connections was explored by Wang [22]. This process was used to highlight differences in how the fully-connected neural network, 1D-CNN, and 2D-CNN extract information from the Flow-Image input. The Segmented-CNN was not evaluated within this section as it does not reveal any additional insight into how the predictions of a neural network

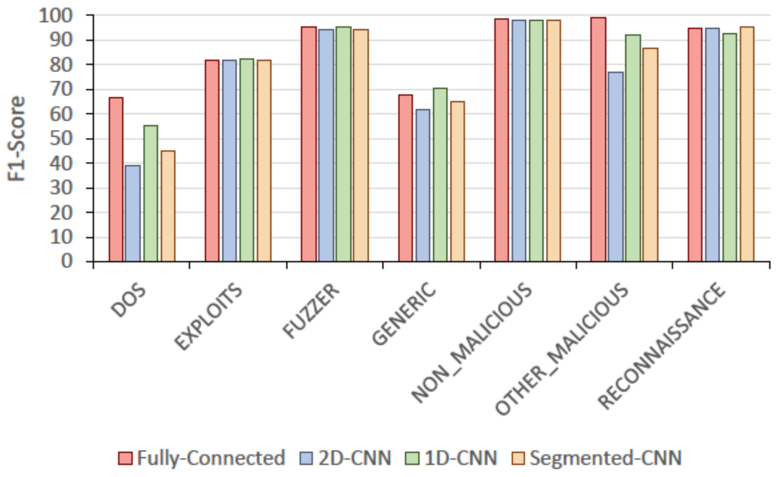


Figure 10. F1-Score of the individual classes for each classifier on the testing set.

are made on the Flow-Image input. A heatmap depicting the most significant inputs to the first fully-connected layer in each neural network is shown in Figure 11.

The heatmap of the first layer in the fully-connected neural network indicates the significance of the byte values in the Flow-Image input. From this heatmap, a highly structured pattern is observed. Distinct patterns were found both in the order of packets in a flow, as well as in the separate header and payload sections of a packet.

Within the header section of the fully-connected heatmap, the utilisation of different header information fields is highlighted. It was shown that the identification field and checksum of the IP header are the two most significant features to the classification of the Flow-Image. These two fields are both unique to their individual packet. It is therefore evident that the neural network is overfitting by learning to identify the individual packets in the training set.

From the payload section of the fully-connected heatmap, it was observed that the neural network identified the first and fourth payload of the flow as the most significant to its classification. The first and fourth payload correspond to the first payload content of a user datagram protocol (UDP) and transmission control protocol (TCP) connection respectively. This result therefore suggests that the first payload content of the flow is most significant to the classification of the overall flow. The correlation between the significance of the payload content and its location within the flow has been confirmed by other investigations [22-24]. This result demonstrates a neural network's ability to

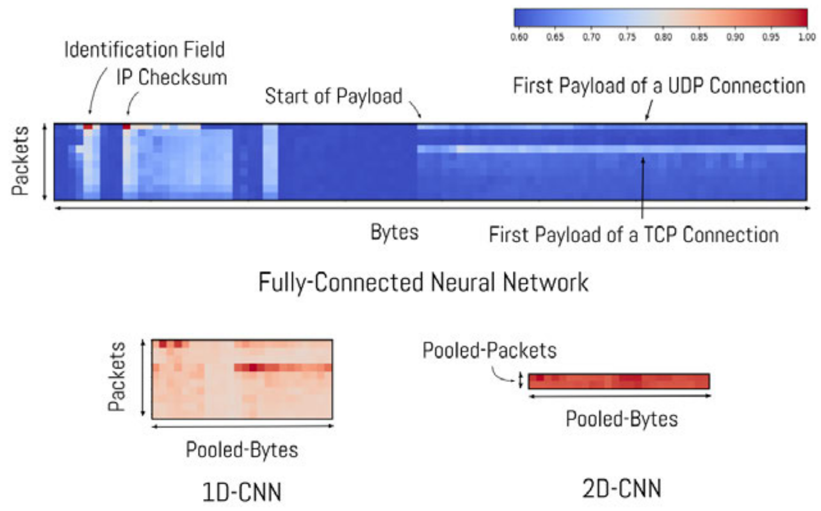


Figure 11. Three heatmaps depicting the most important features in the input of the first fully-connected layer of each architecture. The scale represents a relative importance metric where a value of one indicates the most important feature in the input.

learn identifying properties of complex inputs without any predefined knowledge of the classification domain.

The heatmap of the first fully-connected layer of the 1D-CNN and 2D-CNN shown in Figure 11 indicates the significant features learned by the convolutional and pooling layers of their respective neural networks. From these two heatmaps, we observe how the convolutional and pooling layers can be used to increase the density of important features in the input before the utilisation of computationally expensive fully-connected layers.

The 1D-CNN heatmap illustrates how the one-dimensional convolutional and pooling operations will increase the density of important features of an individual packet while preserving the order of packets in the flow. The 2D-CNN further increases the density of important features in the Flow-Image by also applying the convolutional and pooling operations over the packets in a flow.

The input size of the first fully-connected layer in the 2D-CNN is reduced by a factor of 20 when compared to that of the first layer in the fully-connected neural network. This reduction in the number of input values to the first fully-connected layer vastly reduces the size of the overall network as seen in Section 7.1. The pooling of packets in the flow however decreases the 2D-CNN's ability to learn identifying features from the order in which packets are sent.

7.4 Transient Features

In the previous section, it was highlighted that a neural network learned to classify the network traffic based on assumptions regarding the order of packets in a flow. In this section, an example of how these assumptions could be exploited by malicious evasion techniques will be investigated.

It was identified that the first payload content of a flow is the most significant to its overall classification; however, this assumption cannot always be relied upon, especially when classifying evasive malicious traffic.

Fragmentation is the process of segmenting the content of a traffic flow into multiple packets. Malicious actors have been shown to evade detection by carefully selecting how fragmentation is performed [1]. A simple way to employ such an evasion technique is by purposely sending packets in the wrong order. A host that receives these packets will utilise the packet's sequence numbers to reassemble them into their correct sequence; however, packet reassembly may not be performed within the intrusion detection system (IDS) itself due to the resultant time delay and computational cost. A malicious actor could therefore change the packet sequence to disguise the overall intent of the traffic flow from the IDS [20].

To simulate the use of a fragmentation evasion technique, the order of packets in the Flow-Image input representation was progressively altered. The accuracy of the trained neural networks on the UNSW-NB15 test set was then re-evaluated. A varying number of packets were chosen for re-ordering to identify how much variance in the order of packets would affect the overall traffic classification. The packets that were chosen for re-ordering were random and simply exchanged with the next consecutive packet within the flow. The accuracy of each neural network on the altered test set has been plotted in Figure 12 with respect to the number of packets that were exchanged in each flow.

The classification performance of each neural network decreases as the order of packets is progressively altered; however, classification degradation is less severe in the 2D-CNN and Segmented-CNN. These two neural networks both employ a pooling operation across multiple packets; thereby reducing their reliance on a specific order of packets to correctly classify the flow. This result signifies how CNNs could be implemented to introduce a degree of translation invariance into the classification.

8 Conclusions

In this chapter, the benefits of using a CNN for malicious network traffic classification were explored. A CNN-based approach was shown to achieve similar classification accuracies to a fully-connected neural network at one tenth of the fully-connected neural network's size. This result confirms that a CNN-based solution would be better suited for hardware limited implementations.

The Flow-Image input representation was presented as a novel way to address the issue of using numerical representation for categorical data as seen in the related literature. A numerical representation of byte values in an IDS could be easily exploited as it classifies traffic using properties that may be non-existent in the intended application.

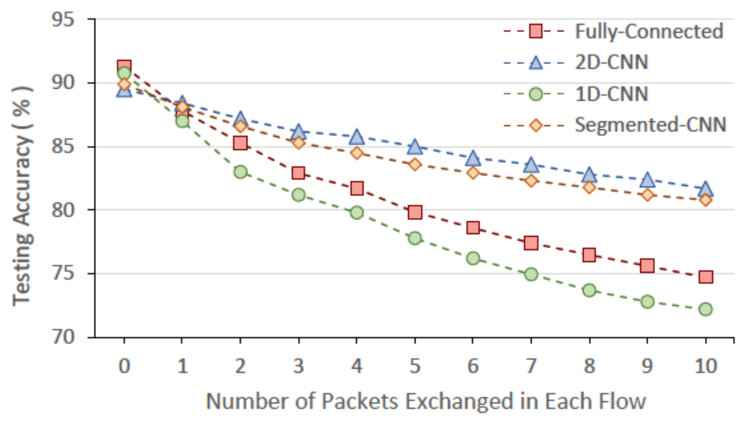


Figure 12. Accuracy of each neural network on the test set with respect to the number of packets that were exchanged in each flow. Packets were exchanged with the next consecutive packet in the flow.

The Segmented-CNN was proposed as a novel CNN architecture that can exploit the distinct sections of a TCP/IP packet, the header and payload. The Segmented-CNN was shown to achieve a faster training time due the use of a divide and conquer approach.

The structure of network traffic and its effect on the classification performance of the evaluated neural networks was identified using a heatmap of the significant input values. The order of packets in a traffic flow was found to be highly influential in the overall classification of a traffic flow.

An example of a simple detection evasion technique was explored by altering the order of packets in a traffic flow. It was shown that the 2D-CNN and Segmented-CNN were more robust to this evasion technique as they employ a pooling operation across multiple packets; thereby reducing their reliance on a specific order of packets to correctly classify the flow.

9 Future Works

In this chapter, the benefits of a CNN to classify noisy data was introduced; however, in network traffic, it is hard to quantify what is noise and what could be a potential malicious threat. Whether this is a benefit or a potential means of evading a CNN based classification algorithm is a subject for further analysis.

Convolutional neural networks require a fixed input size; however, the diversity of network traffic makes it difficult to define what a suitable input size should be. Furthermore, if this input size is known, malicious actors could implement evasion techniques to place malicious code outside the region of analysis. To address this issue, a recurrent neural network (RNN), which allows for variability in the input size, could be combined with a CNN to create a neural network with the mutual benefits of each approach.

The data set used in this chapter for experimental analysis was synthetic. Further evaluation of neural networks on real network traffic captures is required to determine if the performance of these methods hold under real-life conditions.

One of the key contributions of this chapter proposed that CNNs could be employed to counteract certain known types of evasion techniques; however, deep learning techniques can only learn from the existing data they have access to. Techniques such as polymorphic code, encryption, and variations of character encodings would prove difficult to account for without proper consideration. A more robust training method could be implemented by the utilisation of an adversarial network. Specifically, an additional neural network which is trained to produce inputs that evade the detection of the first neural network. Adversarial networks could therefore be investigated as a way to train CNNs such that they are more robust to a broader range of evasion techniques.

References

[1] C. Del Carlo, "Intrusion Detection Evasion: How Attackers Get Past the Burglar Alarm," *SANS Great Lakes, Chicago Illinois,* 2003.

[2] N. Moustafa and J. Slay, "The Significant Features of the UNSW-NB15 and the KDD99 Data Sets for Network Intrusion Detection Systems," in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2015 4th International Workshop on*, 2015, pp. 25-31: IEEE.

[3] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-End Encrypted Traffic Classification With One-Dimensional Convolution Neural Networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2017, pp. 43-48.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems.,* pp. 1097-1105, 2012.

[5] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision,* journal article vol. 115, no. 3, pp. 211-252, 2015.

[6] T. Yoshioka, S. Karita, and T. Nakatani, "Far-Field Speech Recognition Using CNN-DNN-HMM With Convolution in Time," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4360-4364.

[7] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, "Applying Convolutional Neural Networks Concepts to Hybrid NN-HMM Model for Speech Recognition," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 4277-4280: IEEE.

[8] X. Zhang and Y. LeCun, "Which Encoding is the Best for Text Classification in Chinese, English, Japanese and Korean?," *arXiv preprint arXiv:1708.02657,* 2017.

[9] X. Zhang, J. Zhao, and Y. LeCun, "Character-Level Convolutional Networks for Text Classification," in *Advances in neural information processing systems*, 2015, pp. 649-657.

[10] S. Hershey *et al.*, "CNN architectures for large-scale audio classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 131-135.

[11] L. Romaszko, "Signal Correlation Prediction Using Convolutional Neural Networks," in *Neural Connectomics Workshop*, 2015, pp. 45-56.

[12] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2Img: A Sequence-to-Image Based Approach Towards IP Traffic Classification Using Convolutional Neural Networks," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 1271-1276.

[13] H. Zhou, Y. Wang, X. Lei, and Y. Liu, "A Method of Improved CNN Traffic Classification," in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, 2017, pp. 177-181.

[14] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware Traffic Classification Using Convolutional Neural Network for Representation

Learning," in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 712-717.

[15] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Deep Learning for Classifying Malicious Network Traffic," presented at the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Melbourne, Australia, 2018.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *nature,* vol. 521, no. 7553, p. 436, 2015.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[18] Y. LeCun and Y. Bengio, "Convolutional Networks for Images, Speech, and Time Series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, 1995.

[19] V. Dumoulin and F. Visin, "A Guide to Convolution Arithmetic for Deep Learning," *arXiv preprint arXiv:1603.07285,* 2016.

[20] J. A. P. Marpaung, M. Sain, and L. Hoon-Jae, "Survey on Malware Evasion Techniques: State of the Art and Challenges," in *2012 14th International Conference on Advanced Communication Technology (ICACT)*, 2012, pp. 744-749.

[21] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*, 2004, pp. 203-222: Springer.

[22] Z. Wang, "The Applications of Deep Learning on Traffic Identification," *Black Hat USA,* 2015.

[23] G. Aceto, A. Dainotti, W. d. Donato, and A. Pescape, "PortLoad: Taking the Best of Two Worlds in Traffic Classification," in *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, 2010, pp. 1-5.

[24] D. Smit, K. Millar, C. Page, A. Cheng, H. G. Chew, and C.-C. Lim, "Looking Deeper – Using Deep Learning to Identify Internet Communications Traffic " presented at the Australasian Conference of Undergraduate Research (ACUR), Adelaide, 2017.

[25] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature Verification Using a "Siamese" Time Delay Neural Network," in *Advances in Neural Information Processing Systems*, 1994, pp. 737-744.

[26] M. Nour and J. Slay, "UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems (UNSW-NB15 Network Data Set)." *Military Communications and Information Systems Conference (MilCIS),* 2015.

[27] M. Nour and J. Slay, "The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the UNSW-NB15 Sata Set and the Comparison with the KDD99 Data Set.," *Information Security Journal: A Global Perspective,* pp. 1-14, 2016.

[28] A. Martín *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *OSDI*, 2016, vol. 16, pp. 265-283.

Chapter 4

Bipartite Graph Representation for Network Reconnaissance

This chapter provides a novel graph-based point-of-analysis (PoA) for conducting passive network reconnaissance. The provided PoA utilises a bipartite graph to represent and analyse a device's community structure.

Community structure has been widely used in social networking theory to characterise an individual based on their affiliated communities [133]. For example, an individual's political alignment can be revealed through the Facebook groups (i.e., communities) that the individual has joined [134]. In this chapter, we posit that the same community structure holds true for the services (i.e., communities) that devices affiliate with over the Internet.

Analysing a device's community structure would provide the ability to characterise the device through only its affiliated Internet services; thereby, removing the reliance on large heterogeneous feature sets and deep packet inspection (DPI) that conventional ML-based approaches are reliant on (Chapter 3). In contrast, the bipartite PoA provided within this chapter is shown to enable the first known passive network reconnaissance solutions that satisfies all four criteria for widespread deployment within realistic network conditions (CN1-4).

The use of the term *affiliation graph* to define a bipartite graph has been used within the papers presented within this chapter. An affiliation graph is simply the terminology used for a bipartite graph within a social network theory context [133]. The papers within this chapter are included *as is* and therefore their terminology has been unaltered; however, the term bipartite graph was used outside of the included papers to adhere to the more widely accepted terminology.

In total, four papers are presented within this chapter. Each paper utilises the same bipartite PoA; however, each paper provides a distinct analysis into its applicability for network reconnaissance. The four papers presented within this chapter are as follows:

Paper 4.1 - "*Characterising Network-Connected Devices Using Affiliation Graphs*", provides an empirical justification for the use of the bipartite PoA for device characterisation. In particular, the Internet services used by a device were shown through manual, open-source intelligence (OSINT) to reveal the characteristics of the device. We formalise this methodology as a rule-based OS classifier.

Paper 4.2 - "*Clustering Network-Connected Devices Using Affiliation Graphs*", extends a graph clustering technique—stochastic local clustering (SLC) [135]—to infer the composition of devices operating on a network. The technique is qualitatively shown to enable the detection of distinct types of devices such as internal and public facing servers. A quantitative evaluation of the SLC algorithm is provided in Paper 4.3.

Paper 4.3 - "*Detecting Data Exflitration using Seeds based Graph Clustering*", extends the investigation conducted in Paper 4.2 for use in the intrusion detection domain. This paper enables the identification of malicious devices based solely on their community structure. In particular, this paper presents a real-life use-case for the utilisation of community structure to identify malicious devices exfiltrating data from a network environment.

Paper 4.4 - "*Operating System Classification: A Minimalist Approach*", extends Paper 4.1 through the application of conventional ML to automate the analysis of a device's community structure. In particular, a random forest (RF) model was applied to classify the OS of a device based on its community structure. The RF model was shown to achieve a high performance for device characterisation and enables the identification of an OS's community structure to be automated.

The key contribution of this chapter is as follows:

1. (**Paper 4.1**) We present a novel graph-based PoA for network reconnaissance through the representation of a device's community structure as a bipartite graph. The bipartite PoA provides the first comprehensive framework for conducting passive network reconnaissance that relies only on the source and destination IP address fields. This widely available feature set produces network reconnaissance solutions that are scalable, independent of encryption (CN1), and deployable across diverse Internet (TCP/IP) networks (CN2).
2. **(Paper 4.2)** An extension of stochastic local clustering (SLC) was implemented for device characterisation within the bipartite PoA. We show that the devices on a university's network can be largely clustered into five main categories: internal servers, public facing servers, user devices, lab computers, and printers. These clusters would greatly assist network operators in identifying vulnerable or incorrectly configured devices on their networks.
3. **(Paper 4.3)** The extension of SLC implemented in Paper 4.2 was evaluated for intrusion detection within the bipartite PoA. We show that the clusters produced by SLC can be used to identify malicious devices based on their structural similarities to a small set of known malicious devices (i.e., *seeds*). These seeds could therefore be used to identify additional malicious devices that may be operating within a TCP/IP network.
4. (**Paper 4.4**) We show that RF can be used to characterise a device's OS using the bipartite PoA. In particular, it was shown that only 200 communities (i.e., Internet services) are required to identify distinct OSs. Furthermore, we showed that the bipartite PoA is effective for long-term deployment (CN4) on real-world networks (CN3) through analysis of a university campus network over a six-month observation. The solution design within this paper thus provides the first known passive network reconnaissance solution that satisfies all four criteria (CN1-4).

Statement of Authorship

Title of Paper	Characterising Network-Connected Devices Using Affiliation Graphs	
Publication Status	[✓ Published]	Accepted for Publication
	Submitted for Publication	Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Characterising Network-Connected Devices Using Affiliation Graphs," in NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 20-24 April 2020, pp. 1-6.	

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22.

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Characterising Network-Connected Devices Using Affiliation Graphs

Kyle Millar*, Adriel Cheng*,†, Hong Gunn Chew*, and Cheng-Chew Lim*
*School of Electrical and Electronic Engineering, The University of Adelaide, Australia
†Cyber and Electronic Warfare Division, Defence Science & Technology Group, Australia
Email: {kyle.millar, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au*
Email: adriel.cheng@dst.defence.gov.au†

***Abstract*—Device management in large networks is of growing importance to network administrators and security analysts alike. The composition of devices on a network can help forecast future traffic demand as well as identify devices that may pose a security risk. However, the sheer number and diversity of devices that comprise most modern networks have vastly increased the management complexity. Motivated by a need for an encryption-invariant device management strategy, we use affiliation graphs to develop a methodology that reveals key insights into the devices acting on a network using only the source and destination IP addresses. Through an empirical analysis of the devices on a university campus network, we provide an example methodology to infer a device's characteristics (e.g., operating system) through the services it communicates with via the Internet.**

***Index Terms*—Affiliation graphs, device discovery and management, passive network reconnaissance.**

I. Introduction

Device management aims to determine what devices are acting on a network and the purpose of their use. This process helps network administrators provide better quality of service (QoS) and allows security analysts to locate vulnerable devices within their network [3]. However, with the growing number and diversity of devices, implementing an effective device management strategy is increasingly difficult in practice.

A typical enterprise network (such as a university network) is likely to consist of tens of thousands of devices. These devices range from the servers critical to the overall operation of the network, to the personal devices of each user. Additionally, the Internet of things (IoT) paradigm has led to a rapid influx of heterogeneous devices present on such networks (e.g., printers, IP cameras, and smart sensors) [11]. The rising complexity of providing adequate device management has not only hindered the administration of enterprise networks but has also introduced security vulnerabilities due to unknown and unpatched devices [2]. It is therefore evident that methods of reducing the complexity of device management is of critical importance.

Current techniques of device management have typically relied on either a large number of difficult to extract features (e.g., Internet protocol (IP) ID monotonicity and clock frequency) or target a specific subset of devices [6]. The wide array of devices present on modern networks renders these techniques either computationally ineffective at scale or are not universally applicable. It becomes necessary to infer information about the devices on a network through a limited but universally applicable feature set.

Affiliation graphs are a social networking technique that investigates the characteristics of an individual through the communities they are affiliated with [5]. Social communities form through both implicit and explicit commonalities in their membership set; such as shared interests, occupation, or genealogy [14]. Through analysing the shared commonality within a community, certain characteristics of its membership set can often be revealed. For example, Facebook is able to identify the political alignment of a user based solely on the Facebook groups the user is affiliated with [9].

The advantage of an affiliation graph lies in the ease in which the affiliations between an individual and their communities can be observed [4]. Through understanding why these affiliations are made, interesting inferences can be made about an individual without requiring any *a priori* knowledge of the individual themselves.

It is the axiom of this work that certain device characteristics (e.g., manufacturer, operating system (OS), and installed applications) can be determined by the communities a device affiliates with via the Internet. For example, a device affiliated with a Windows update server is likely to be running the Windows OS [6]. Through extrapolating this approach to additional communities with known commonalities, a profile of the device's overall characteristics could therefore be constructed.

Investigating the communities that a device is affiliated with rather than the device itself is useful for two key reasons. First, the characteristics of an unknown set of devices can be inferred from a known set of devices given they exhibit a similar structural equivalence.[1] Second, it allows for the identification of a device's characteristics without requiring direct analysis of the device itself. This is useful in various scenarios such as when the device is only intermittently connected to the

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

[1]In an affiliation graph, the structural equivalence of two devices is proportional to the number of communities they share.

network (e.g., a mobile device) or the device is no longer present on the network at all.

In our study, we examine the relationship between devices on a network (*actors*) and the destination addresses (*communities*) in which these devices communicate with via the Internet. This simplifies our analysis to just two easily obtainable fields – the source and destination IP addresses. In restricting our analysis to just these two fields, we create an encryption invariant basis of analysis that is quick to populate and requires little storage. An example of constructing an affiliation graph from the source and destination IP addresses is shown in Figure 1.

The key contribution of our work is twofold: (1) To the best of our knowledge, this is the first work which identifies a device's characteristics through analysis of the destination IP addresses (*communities*) it communicates with. The examination of a device's affiliated communities, rather than the device itself, provides a method of device characterisation that is useful when the device itself cannot be examined; and (2) we provide an empirical analysis of our technique on a university campus network. In particular, we provide an example of how this method could be implemented to determine a device's OS.

The remainder of this paper is structured as follows: A brief survey of related work in device management is given in Section II. Section III outlines the process of creating an affiliation graph from two 26-hour captures of a university's network. Analysis of an affiliation graph's community set in relation to network traffic is explored in Section IV. Section V provides an empirical example of how the analysis of this community set can give insight into the operating systems of the devices on a network. The paper concludes in Section VI with a discussion of results and suggestions for future work.

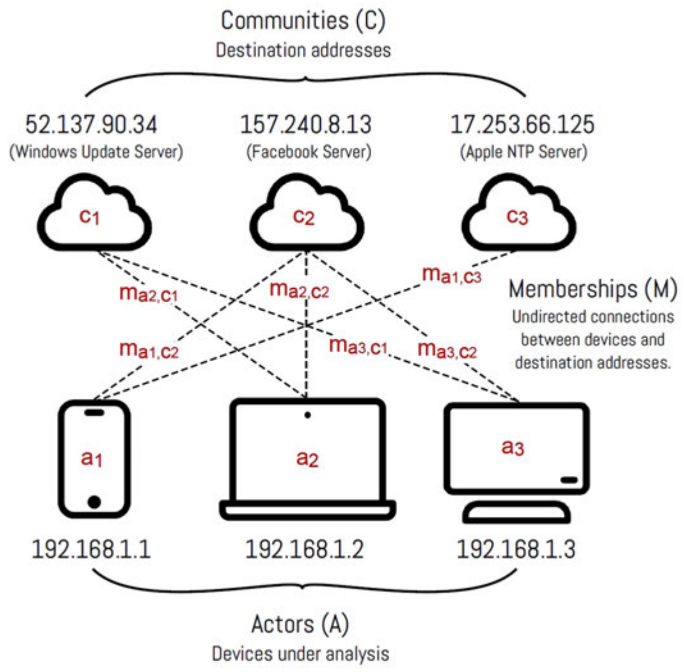


Fig. 1. An affiliation graph created from network data. The actors and communities of the affiliation graph are the devices internal and external to the network under analysis, respectively. The membership set consists of the network communications between the internal and external devices.

II. Related Work

There has been growing research interest in device management strategies due to the increase in the number of devices with Internet connectivity; however, device management is still underrepresented by the research community at large.

Arora et al. [3] highlighted the security benefits of identifying key devices on a network. The knowledge of these key devices and their typical characteristics can help to assess and mitigate the impact in the event of a security breach.

Arora et al. went on to investigate techniques to identify the servers present on a network. Although achieving high detection rates using machine learning techniques, several feature extraction stages were first required. The computational requirement of these feature extraction stages reduces the real word efficacy of this method for any moderate sized network. Furthermore, the identification of servers alone is insufficient to effectively administer a network. Increasingly, bring-your-own device (BYOD) policies are causing issues in the management of enterprise networks [13].

BYOD policies have been widely introduced in educational and business networks such that their users can connect their personal devices to the network. While this is convenient for users of the network, it introduces novel challenges in how to manage and secure a 'user device' for which the network administrators have little control over.

IoT devices have also posed serious concerns for user privacy and security within an enterprise network [11]. Methods to mitigate these concerns have sparked interest in device management strategies to identify and manage IoT devices operating on a network [8]; however, the current state of IoT device management has been complicated due to the heterogeneity and rapid evolution of such devices.

In this paper, we introduce a novel method of device characterisation through the use of affiliation graphs. Although there has been speculation that a device's characteristics can be inferred through the services for which it connects to via the Internet [6], [13], this paper provides a methodology to extract these relationships. In particular, our method aims to address the dynamic nature of device characterisation by utilising only two universally applicable features – the source and destination IP addresses.

III. Affiliation Graphs for Device Management

The axiom of affiliation graphs is that there exists a reason for which actors are affiliated with certain communities [14]. For example, Internet-connected devices will either contact or be contacted by certain servers to synchronise the services running on the device (e.g., Facebook notifications). This insight suggests that the client-side services of a device could be inferred through their affiliation to their serve-side counterpart.

A. Notation

An affiliation graph is a bipartite graph consisting of two distinct sets of vertices - actors (A) and communities (C) [5]. An actor ($a \in A$) is said to be affiliated with a community ($c \in C$) if there exists a membership ($m_{a,c} \in M$) between a and c; where M is the set of all memberships between A and C. An affiliation graph can therefore be represented as the collection of actor, community, and membership sets, $G = (A, C, M)$.

B. Populating an Affiliation Graph from Network Data

The benefit of an affiliation graph is the ease in which it can be constructed. To create an affiliation graph from network data, only the source and destination IP addresses are required from each network flow[2]. This results in a technique that is quick to populate and requires little storage. Additionally, this technique is also widely deployable as most network administrators typically have access to such flow records.

The pseudo-code for creating an affiliation graph from network data is provided in Algorithm 1. It should be noted that this method utilises a weighted affiliation graph such that the weight of a membership, $\omega(m)$, is the number of flows between a device and a community.

C. University Campus Data Set

To analyse the efficacy of applying affiliation graphs on a large and diverse network, two captures of the University of Adelaide's enterprise network were taken. Both captures consisted of 26 hours of network traffic for which all flow records were recorded. The first capture, named *low-activity*, was taken on Christmas day in 2018 and captured approximately 19 million flow records. This capture was scheduled to ensure that as few devices were active on the network as possible and therefore serves as a baseline profile of the network. The second capture, named *high-activity*, was taken on the 1st of May 2019[3] and captured approximately 33 million flow records. This capture was taken during a typical workday where the network was actively being used by both staff and students.

The *low-activity* affiliation graph took approximately half an hour to populate; whereas, the *high-activity* affiliation graph took just under an hour. The time taken to create both affiliation graphs highlights the ability of this method to scale well even when the activity on a network increases significantly. Table I provides a summary of the affiliation graphs created from the two network captures. From this table, the following insights were identified:

1. The number of devices present on the *high-activity* capture is significantly greater than that of the *low-activity* capture. This result is consistent with the increase in users on the network during the *high-activity* capture.
2. Less than 15% of flows within both captures have a unique set of source and destination IP addresses. This result shows that the majority of traffic generated or received by a device consists of repeated connections to certain communities.

A device's affiliation to a given community can give insight into the device itself. For example, the devices affiliated with the community '210.173.216.59' within the *high-activity* affiliation graph are depicted in Figure 2. This community is owned by 'Ricoh Company, Ltd.', a company most notably known for manufacturing cameras and printers. It was found that of the 90 devices affiliated with this community, 89 were printers manufactured by the Ricoh company. It is therefore likely that an additional device (new or previously unknown) which affiliates with this community would also be a Ricoh printer. This example illustrates the ability to draw information about a device through their affiliated communities. In the next section, techniques to extract useful information about the community set are examined.

Algorithm 1 Populating an Affiliation Graph

```
1: for flow in flow records do
2:     Record source and destination IP of flow.
3:     Set device, $a$, as the IP address within the network
       under analysis.
4:     Set community, $c$, as the IP address outside the
       network under analysis.
5:     Denote the membership between $a$ and $c$ as $m_{a,c}$.
6:     If $a \notin A, A = A \cup \{a\}$
7:     If $c \notin C, C = C \cup \{c\}$
8:     If $m \notin M; \omega(m_{a,c}) = 1, M = M \cup \{m_{a,c}\}$; else,
       $\omega(m_{a,c}) = \omega(m_{a,c}) + 1$
9: end for
where $\omega(m)$ is the weight of the membership $m$.
```

[2]A network flow (or flow) is defined in this work as the bi-directional exchange of information between two devices over the Internet.

[3]The 1st of May is not a public holiday in the location in which the capture took place.

IV. Community Analysis

A unique property of an affiliation graph created from network data is the wealth of information that can be gathered about the community set. Through utilising standard network reconnaissance tools such as WHOIS [7] and the reverse domain name system (rDNS) [10], the service that a community provides can often be determined. This examination of the community set can provide a useful technique of inferring information about a device even when the examination of the device itself is infeasible (e.g., when the device is no longer active on the network under analysis).

TABLE I
Affiliation graphs summary statistics. $|A|$ is the number of devices, $\mu_{deg(A)}$ is the mean number of communities per device, $|C|$ is the number of communities, $\mu_{deg(C)}$ is the mean number of devices in each community, and $|M|$ is the number of memberships.

Data Set	$\|A\|$	$\mu_{deg(A)}$	$\|C\|$	$\mu_{deg(C)}$	$\|M\|$
low-activity	6.1k	201	476k	2.55	1.22M
high-activity	24.7k	158	566k	6.89	3.90M

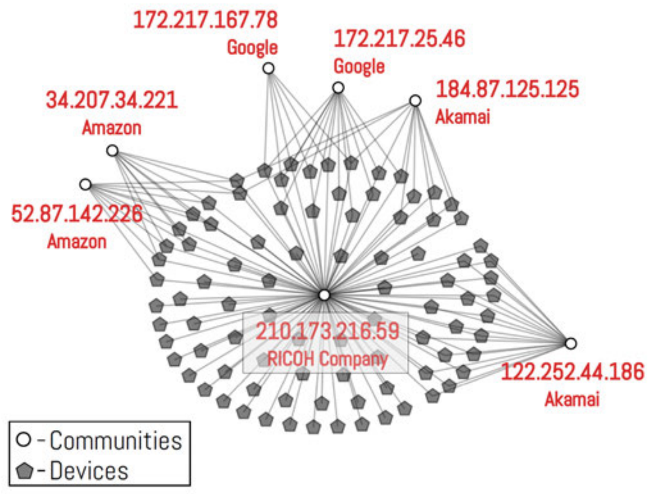


Fig. 2. A subset of the *high-activity* affiliation graph. This subset depicts the devices affiliated with the community '210.173.216.59'. This community is owned by 'Ricoh Company, Ltd.'. 89 of the 90 devices affiliated with this community were found to be printers manufactured by Ricoh itself.

It is worth noting that the analysis of the community set can be done independently from the analysis of the target network itself. This allows community analysis to be evaluated in an environment for which the examiner has complete control.

The service a community provides can give insight into the reason for which a device is affiliated with a certain community. For example, the community '157.240.8.35' can be resolved to one of Facebook's Australian servers. A device with a membership to this community is therefore likely (1) located in Australia, (2) running a Facebook service, and (3) a user device. Additionally, the device is also likely to be a mobile device as it has been found that 79% of social media is now accessed through mobile devices [1]. A profile of the device's overall characteristics (e.g., manufacturer, OS, and installed applications) could then be created through extrapolating this approach to additional known communities.

To investigate what device characteristics could be inferred from community properties; the (1) ownership, (2) domain name, and (3) associated application traffic of each community were resolved where possible. From these community properties, the service a community provides could then be estimated.

1) Ownership – The ownership of an IP address is the registered entity responsible for that IP address. The ownership of a community can provide an overview of the types of services the community provides. For example, a community owned by Facebook will likely host one of Facebook's social networking services (i.e., Whatsapp, Instagram, and Facebook). While the exact service of the community cannot be determined in this scenario, it reduces the search space for the potential services that are likely to be offered by this community. Additionally, in the case where the registered owner only provides a singular service (e.g., Twitter), the ownership of an IP address can directly signify the purpose of its respective community.

The ownership of an IP address can be resolved through a WHOIS query. A WHOIS query will return information about the registered entity responsible for the queried IP address. This information can then be used to determine what services are likely to be associated with this community.

The ownership of a community however is not always useful for the purposes of our analysis. For example, content distribution networks (CDNs) provide many distinct services under a singular ownership. Therefore, further information about such communities would be required to determine their services.

2) Domain Name – A device's domain name is a human readable name associated with the device. Often a device's domain name is created using a particular naming convention which can provide useful hints to the service the device provides. For example, the domain name of the community '17.253.66.125' was resolved to '$ausyd2$-ntp-$001.aaplimg.com$' using rDNS. This domain name suggests that the community is (1) located in Sydney, Australia ($ausyd$); (2) a network time protocol (NTP) server (ntp); and (3) owned by Apple ($aaplimg.com$[4]). Therefore, the service of this community is likely to be providing accurate time resolution to Apple devices within the Australia region. It should be noted however that the domain name of a community within a CDN is unlikely to resolve further information about the community itself.

The main purpose of a CDN is to provide distributed hosting of popular Internet applications (e.g., YouTube, iTunes, and Twitter) [12]. The service provided by a CDN community is therefore closely associated with the application it hosts. The service of a community CDN can therefore often be inferred through analysis of its associated application network traffic.

3) Application Traffic – The application traffic associated with a given community can be a strong indication of the service that community provides. For example, 97% of the traffic to and from the community '74.125.130.109' was found to be associated with Google's mail application, Gmail. Therefore, it can be assumed that a device affiliated with this community is using the Gmail application.

The application traffic associated with each community was identified through correlating the application logs taken during the capture period. These application logs were generated through a combination of deep packet inspection and metadata analysis. Resolving the application traffic associated with a community can only be achieved when there exists a system of classifying application traffic on the network; however, once the service of a community is identified it can be used in subsequent analysis without requiring its properties to be resolved again.

An example of resolving all three community properties for three example communities is given in Table II. It is seen that multiple correlating sources of information allow for a strong indication of the service that a community offers.

The community properties resolved within this section are likely to be the reason for which a device would affiliate with

[4]The ownership of the *aaplimg.com* domain is owned by Apple, Inc. as resolved through a WHOIS query

TABLE II
IDENTIFIED PROPERTIES OF THREE EXAMPLE COMMUNITIES.

Community Example	IP Address	Ownership	Domain Name	Application Traffic	# *low-activity* devices	# *high-activity* devices
#1	157.240.8.35	Facebook, Inc.	edge-star-mini-shv-01-syd2.facebook.com	Facebook (95%), Incomplete (3%), SSL (2%)	322	$9,799$
#2	17.253.66.125	Apple, Inc.	ausyd2-ntp-001.aaplimg.com	NTP (100%)	235	$1,617$
#3	74.125.130.109	Google, Inc.	sb-in-f109.1e100.net	Gmail (97%), Incomplete (3%)	56	$2,389$

a specific set of communities. For example, a device affiliated with the Facebook community in Table II is presumably running the Facebook client-side application. The utilisation of these inferences allows for information about the set of devices to be resolved through analysing the communities they affiliate with.

V. Device Analysis

This section serves as a preliminary example of how the community analysis in Section IV can be applied to infer the characteristics of the devices operating on a network. The example provided in this section examines the use of an affiliation graph to identify the operating systems (OSs) of a device. Identifying the OS of the devices on a network is critical for providing quality of service and assessing the security vulnerabilities of the network as a whole. As such, OS classification is an important area of interest in device characterisation.

The axiom of identifying the OS of a device through an affiliation graph is that there is an intrinsic relationship between the device's OS and the communities it affiliates with. Intuitively, this axiom seems plausible given a device will often connect to services related to its operating system (e.g., to check for new notifications and software updates). This intuition is illustrated in Figure 3 where the community affiliations of eight devices in the *high-activity* capture were examined. It is seen that a large portion of the affiliations of each device are to the maintainers of their respective OS.

To investigate the use of an affiliation graph to distinguish between distinct OSs, a subset of known devices from the high-activity capture were analysed. In total, $10,521$ devices were selected for analysis as their operating systems could be validated through either auxiliary logs or via physical inspection. The distribution of operating systems under analysis in this section is shown in Table III.

The following subsections investigate whether a particular OS can be inferred through only evaluating its affiliated communities. A summary of the results found for Apple OS and Windows OS can be seen in Figure 4.

A. Apple OS

A strong relationship was found between devices running an Apple OS[5] and the communities owned by Apple itself. Given the set of devices running an Apple OS, $5,239$ (95.7%) were affiliated with five or more Apple owned communities. Conversely, of the $5,285$ devices that affiliated with five or more Apple communities, 99.1% were a device running an Apple OS. This result indicates a significant dependency on Apple's online ecosystem to manage the client-side services running on Apple's OSs. Furthermore, due to Apple's exclusivity in offering services for only its own products, it proves trivial to separate an Apple OS from other OSs through only resolving the ownership of its affiliated communities[6]. Further investigation however would be required to distinguish between devices running iOS and OS X using this method.

B. Windows OS

Through investigating the ownership of the communities alone there proved to be no significant relationship between communities owned by Microsoft and devices running Windows OS. This result is not unexpected as Microsoft provides a wide

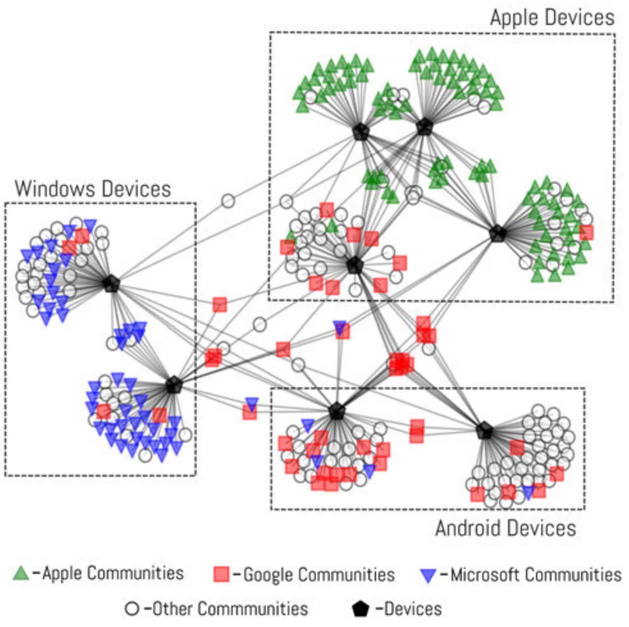


Fig. 3. A subset of the *high-activity* affiliation graph depicting the community affiliations of eight devices with known characteristics. Four devices which were running an Apple OS (three iOS, one OS X), two devices running Windows OS, and two devices running the android OS. For clarity, only the top 50 most heavily weighted community affiliation of each device is depicted.

[5]Apple OS has been defined as either Apple's mobile OS (iOS) or desktop OS (OS X).

[6]provided the device is exhibiting typical network behaviour.

TABLE III
DISTRIBUTION OF OPERATING SYSTEMS IN THE KNOWN SUBSET OF THE HIGH-ACTIVITY AFFILIATION GRAPH.

Operating System	Number of devices
iOS	3,872 (36.80 %)
OS X	1,602 (15.23 %)
Windows	3,285 (31.22 %)
Android	1,762 (16.75 %)
Total	$10,521$ (100.0 %)

range of services that are not only limited to devices running Windows OS (e.g., Office365). Therefore, a greater focus was placed on the service the community provides rather than its ownership alone.

From the analysis conducted in Section IV, 122 communities offering Windows specific services were found $-$ 111 providing Windows updates and 11 providing the Windows push notification service (WNS). Of the 1,101 devices with an affiliation to at least one of these communities, $2,454$ (97.3%) were confirmed to run Windows OS. Additionally, 74.7% of the $3,285$ devices running a Windows OS were identified to affiliate with one or more of these communities. This result indicates that there is a relationship between communities offering Windows related services and devices running Windows OS; however, it is not as comprehensive as the relationship between Apple OS and Apple based communities.

C. Android OS

No distinct set of Google communities were found to result in a significant relationship with devices running Android OS. This result is due to the myriad of OS-agnostic services that offered by Google (e.g., Googles' search engine, Chrome, and YouTube). Furthermore, as Android is maintained by many distinct entities it proves more difficult to determine a specific set of communities that would be representative of its online services.

A potential method to increase the information contained in the affiliation graph would be to differentiate the community set not only by an IP address but also by port number. This would allow the analysis to distinguish between distinct services running on the same IP address.

VI. CONCLUSION AND FUTURE WORK

In this paper, the application of affiliation graphs for characterising devices on an enterprise network was evaluated. A methodology for analysing the community set was investigated which allowed for inferences to be made about the characteristics of a device through the communities it affiliates with. An example was given to demonstrate the applicability and effectiveness of applying an affiliation graph to identify the OSs of the devices on a network. Overall, it was found that an affiliation graph is a quick and easy method to gain an initial insight into the characteristics of the devices on a network. However, further analysis should be conducted to identify robust relationships between devices characteristics and their affiliated communities.

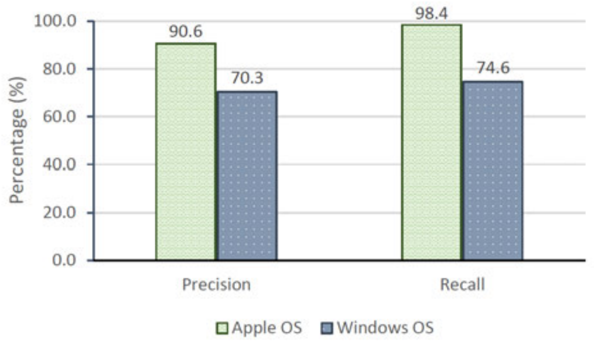


Fig. 4. Precision and recall of Apple and Windows devices found through investigating their community set. Apple devices are defined as a device running either iOS or Mac OS; whereas, Windows devices are defined as a device running the Windows OS.

In future work we aim to provide a broader analysis on utilising affiliation graphs for device characterisation. In particular, we aim to utilise supervised machine learning techniques to help identify the relationship between a device's characteristics and the communities it affiliates with.

REFERENCES

[1] The global mobile report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-Global-Mobile-Report, 2017. [Online] Accessed on: September 2019.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[3] Deepali Arora, Kin Fun Li, and Alex Loffler. Big data analytics for classification of network enabled devices. In *Advanced Information Networking and Applications Workshops (WAINA)*, pages 708–713, 2016.

[4] Stephen Borgatti and Daniel Halgin. Analyzing affiliation networks. *The Sage Handbook of Social Network Analysis*, 2011.

[5] Ronald L Breiger. The duality of persons and groups. *Social Forces*, 53(2):181–190, 1974.

[6] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. OS fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 173–180. ACM, 2014.

[7] Leslie Daigle. RFC-3912 WHOIS protocol specification. *Network Working Group*, 2004.

[8] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized IoT devices using machine learning techniques. *arXiv preprint arXiv:1709.04647*, 2017.

[9] Jeremy Merrill. Liberal, moderate or conservative? see how Facebook labels you. https://www.nytimes.com/2016/08/24/us/politics/facebook-ads-politics.html, 2016. [Online] Accessed on: September 2019.

[10] Paul Mockapetris. RFC-1035 domain names-implementation and specification. *Network Working Group*, page 23, 1987.

[11] Bruce Schneier. The internet of things will upend our industry. *IEEE Security & Privacy*, 15(2):108–108, 2017.

[12] Athena Vakali and George Pallis. Content delivery networks: status and trends. *IEEE Internet Computing*, 7(6):68–74, Nov 2003.

[13] Xuetao Wei, Nicholas C Valler, Harsha V Madhyastha, Iulian Neamtiu, and Michalis Faloutsos. Characterizing the behavior of handheld devices and its implications. *Computer Networks*, 114:1–12, 2017.

[14] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *IEEE 12th International Conference on Data Mining*, pages 1170–1175. IEEE, 2012.

Statement of Authorship

Title of Paper	Clustering Network-Connected Devices Using Affiliation Graphs
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, ``Clustering Network-Connected Devices Using Affiliation Graphs," in International Conference on Machine Learning and Cybernetics (ICMLC), 2021: IEEE, 2021, pp. 1-6.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Clustering Network-Connected Devices Using Affiliation Graphs

Kyle Millar[1], Adriel Cheng [1,2], Hong Gunn Chew[1], Cheng-Chew Lim[1]

[1]School of Electrical and Electronic Engineering, The University of Adelaide, Australia
[2]Cyber and Electronic Warfare Division, Defence Science & Technology Group, Australia
E-MAIL: {kyle.millar, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au[1]
E-MAIL: adriel.cheng@dst.defence.gov.au[2]

Abstract:

Device management in large networks is of growing importance to network administrators and security analysts alike. The composition of devices on a network can help forecast future traffic demand as well as identify devices that may pose a security risk. However, the sheer number and diversity of devices that comprise most modern networks has vastly increased the complexity of performing this management. Motivated by these issues, we examine the application of affiliation graphs to quantify the relationship between devices operating on a network and the services for which they connect to via the internet. These relationships can then be used to identify clusters of devices which exhibit similar behavioural characteristics.

Through empirical analysis of two 26-hour captures of a university campus network, we show that affiliation graphs can be utilised to cluster the devices on a network without any *a priori* knowledge of the network itself. In particular, our preliminary results show that devices can be clustered into specific device types (e.g., servers, user devices, and printers). These clusters can then be used to examine the composition of devices on the network, create informed device management policies, and identify potentially vulnerable devices.

Keywords:

Affiliation graphs; cybersecurity; clustering; device management

1. Introduction

Device management is a critical component of network administration which aims to determine what devices are acting on a network and the purpose of their use. This process helps network administrators provide better quality of service (QoS) and allows security analysts to locate vulnerable devices within their network [2]. However, implementing an effective device management strategy is becoming increasingly difficult in practice.

The fundamental challenge of device management is in the growing number and diversity of devices acting on modern networks. A typical enterprise network (such as a university network) is likely to consist of tens of thousands of devices. These devices range from the servers critical to the overall operation of the network, to the personal devices of each user. Additionally, the internet of things (IoT) paradigm has led to a rapid influx of heterogeneous devices present on such networks (e.g., printers, IP cameras, and smart sensors) [15].

The rising complexity of providing adequate device management has not only hindered the administration of enterprise networks but has also introduced security vulnerabilities due to unknown and unpatched devices [1]. It is therefore evident that methods of reducing the complexity of device management is of critical importance.

Current techniques of device management have typically relied on either a large number of difficult to extract features (e.g., internet protocol (IP) ID monotonicity and clock frequency) or target a specific subset of devices [5]. However, these techniques are either becoming computationally ineffective at scale or are not universally applicable to manage the wide array of devices present on modern networks. Motivated by these issues, we pose the use of affiliation graphs to infer information about the devices on a network through only the IP addresses that they affiliate with over the internet.

Affiliation graphs are a social networking technique that investigate the characteristics of an individual through the communities they are affiliated with [4]. Social communities form through both implicit and explicit commonalities in their membership set; such as shared interests, occupation, or genealogy [18]. Through analysing the shared commonality within a community, certain characteristics of its membership set can often be revealed. For example, Facebook is able to identify the political alignment of a user based solely on the Facebook groups the user is affiliated with [10].

It is the axiom of this work that a behavioural profile of a device can be constructed through the examination of the communities that the devices affiliates with via the internet. The communities in this work are defined by online services such as update servers, cloud storage, and software as a service (SaaS) distributions. The rising reliance on online (*cloud*) services therefore allows a profile of a device to be constructed without directly investigating the device itself. Investigating the communities that a device is affiliated with, rather than the device itself, is particularly useful when the device is no longer present on the network under analysis.

Notation. An affiliation graph is a bipartite graph consisting of two distinct sets of vertices - actors (A) and communities (C) [4]. An actor $(a \in A)$ is said to be affiliated with a community $(c \in C)$ if there exists a membership $(m_{a,c} \in M)$ between a and c; where M is the set of all memberships between A and C. An affiliation graph can therefore be represented as the collection of actor, community, and membership sets, $G = (A, C, M)$.

The advantage of an affiliation graph lies in the ease in which the affiliation between an actor and their communities can be observed [3]. Through understanding why these affiliations are made, interesting inferences can be made about the actor without requiring any *a priori* knowledge of the actor themselves.

In our study, we examine the relationship between devices on a network (*actors*) and the destination addresses (*communities*) in which these devices communicate with via the internet. This simplifies our analysis to just two easily obtainable fields – the source and destination IP addresses. In restricting our analysis to just these two fields, we create an encryption invariant basis of analysis that is widely deployable to distinct network configurations. An example of constructing an affiliation graph from the source and destination IP addresses is shown in Figure 1.

We further this study by examining the similarity between the devices on a network based on their mutual community affiliations. This similarity metric allows for clusters of similarly behaving devices to be found on a network. Identifying clusters of devices on a network aids in the creation of informed device management policies and allows for the identification of potentially vulnerable devices [8].

The contribution of this work is two-fold:

1. To the best of our knowledge, this is the first work utilising affiliation graphs to cluster the devices operating on a network. We show that an affiliation graph can be used to quantify the similarity between the devices on a network and hence identify clusters of devices that share common characteristics.

2. We validate our technique through empirical analysis of two 26-hour captures of a university campus network. Our preliminary results show that our technique is able to cluster the devices on the network under analysis into five main categories: internal servers, public facing servers, user devices, lab computers, and printers.

The remainder of this paper is structured as follows: A brief survey of related work in affiliation graphs and device management is given in Section 2. Section 3 introduces the proposed methodology for clustering the devices on a network. Section 4 outlines the dataset used for our analysis. Section 5 provides an empirical analysis of the efficacy of the proposed technique. The paper concludes in Section 6 with a discussion of results and suggestions for future work.

2. Related Work

Sociology has long since recognised the importance of analysing social affiliations between individuals and the communities for which they belong [16]. Social affiliations can help to define a person's interests [4], community relationships (e.g., family, friends, and colleagues) [18], and can also help to recommend new affiliations [19]. However, this is the first study to investigate whether these same affiliations can be used to cluster similarly behaving devices on a network.

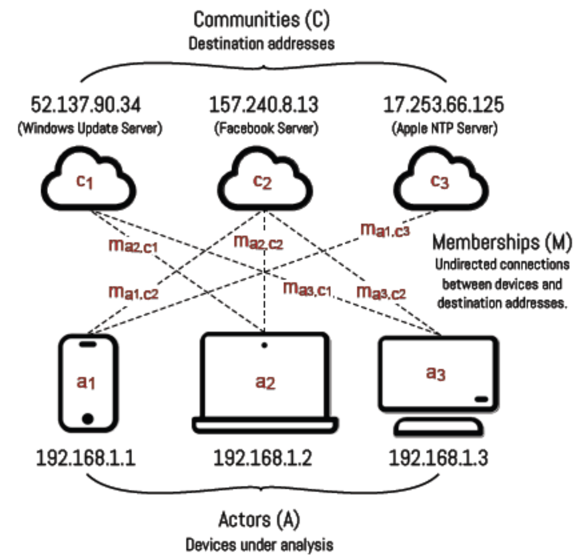


FIGURE 1. An affiliation graph created from network data. The actors and communities of the affiliation graph are the devices internal and external to the network under analysis, respectively. The membership set consists of the network communications between the internal and external devices.

There has been growing research interest in device management strategies due to the increase number and diversity of devices present on modern networks; however, device management is still under-represented by the research community at large. Research thus far has typically focused on examining the role of a particular type of device on a network; identifying its unique characteristics and its effect on the overall management of the network. Devices types such as servers [2, 8], user devices [17], and IoT devices [9] have been of particular interest in such studies. However, to the best of our knowledge, there has not yet been a study on a method of clustering unknown types of devices that are present on a network.

In our previous work [12, 11], we have shown the feasibility of classifying the operating system of a device through the use of affiliation graphs. In this paper, we provide an extension of this work to cluster similarly behaving devices on a network. We show that certain device types (e.g., servers, user devices, and printers) have distinctive behavioural characteristics that can be identified through the services that they connect with via the internet.

3. Method

The behaviour of a device on a network is seldom unique in its entirety. Devices share similar characteristics due to the commonality in their manufacturers, operating systems, the services they run, or even their users. An affiliation graph can be used to examine the similarity between two devices based on the commonality of the communities that they frequent. For example, two devices running Apple's mobile operating system (iOS) are likely to frequent similar communities to check for new updates, backup files using iCloud, and to check for new notifications using the Apple push notification service (APNS). Investigating the overlap in their community sets allows for a metric of the overall similarity between these two devices to be defined. This similarity metric can then be used to cluster the devices into distinct behavioural groups.

Salton's cosine coefficient [13] was used to quantify the similarity between the actors in the affiliation graph. Salton's cosine coefficient states that the similarity between two vectors, v_1 and v_2, is the cosine of the angle between them. The similarity between two actors, a_1 and a_2, can therefore be found by computing Salton's cosine coefficient of their membership sets as shown in Eq (1).

$$\text{sim}(a_1, a_2) = \frac{M(a_1) \cdot M(a_2)}{||M(a_1)|| \times ||M(a_2)||} \tag{1}$$

$$M(x) = \{\omega(m_{x,c}) : c \in C\}, \text{if } m_{x,c} \notin M; \omega(m_{x,c}) = 0$$

where $M(x)$ is the membership vector of an actor, x, and, $\|M(x)\|$ is the length of the membership vector, $M(x)$. As $M(x) \geq 0$, the similarity between two actors is bounded between $[0, 1]$. A similarity of 0 indicates two actors with no intersection of their community sets; whereas, a similarity of 1 indicates two actors who have affiliated with the same set of communities, the same number of times throughout the capture period.

Evaluating Salton's cosine coefficient of each pair of actors results in the complete, homogeneous graph, $G_A = (A, E)$. Where, E, is the set of all similarities between the actors under analysis, A. To reduce the density of this graph, a similarity threshold, τ, was introduced. This threshold removed all edges from the graph with a similarity $\leq \tau$. A similarity threshold of $\tau = 0.8$ was chosen to maximise the global clustering coefficient of the resultant graph. The resultant graph produced by this method was denoted as a *similarity graph*.

Stochastic local clustering (SLC) [14] was applied to the similarity graph to find distinct clusters of similarly behaving devices. The insight behind SLC lies in the ability to find a cluster for an initial vertex, v, through a stochastic search of its local connections. This process is denoted as the local search of v. The full SLC algorithm is the combinatorial analysis of applying local search for each vertex in the graph. The utilisation of local search allows SLC to be scalable to significantly large analyses as the full graph never needs to be considered in its entirety. Furthermore, SLC is inherently parallelisable given that local search for distinct vertices can be preformed independently.

An overview of the SLC algorithm used to obtain the results in Section 5 has been outlined in the Appendix.

4. Dataset

To evaluate the proposed methodology, two 26-hour captures of the [*university*]'s campus network were taken. The first capture, *low-activity*, was taken on Christmas day in 2018. This capture was scheduled to provide an analysis of the university's network when minimal user-related activity would be observed. The second capture, *high-activity*, was taken on the 1st of May 2019. This capture was scheduled to provide an analysis of the university's network on a typical work day[1].

The graph statistics for both captures are provided in Table 1. It is seen that there is a significant increase in network activity between the *low-* and *high-activity* captures. This is largely due to an influx in the number of user-devices on the network during the *high-activity* capture.

[1]The 1st of May is not a public holiday in the location in which the capture took place.

TABLE 1. Summary statistics of the affiliation and similarity graphs created from the two network captures. $|A|$ is the number of devices, $\mu_{deg(A)}$ is the mean number of communities per device, $|C|$ is the number of communities, $\mu_{deg(C)}$ is the mean number of devices in each community, $|M|$ is the number of unique memberships, and $|E|$ is the number of edges in the resultant similarity graph.

Data Set	$\|A\|$	$\mu_{deg(A)}$	$\|C\|$	$\mu_{deg(C)}$	$\|M\|$	$\|E\|$
low-activity	4.9k	102	78.8k	7.5	0.6M	0.6M
high-activity	13.5k	84	86.9k	14.7	1.3M	3.2M

5. Results

The result of applying the SLC algorithm to the *low-* and *high-activity* similarity graphs is shown in the ordered adjacency matrices in Figure 2. It is seen that distinct clusters of devices were found in both captures of the university's network. These clusters signify that similarly behaving devices on a large network can be easily identified through only the communities that the devices affiliate with via the internet. Therefore, the use of an affiliation graph would provide a significant basis of analysis for network operators to investigate the groups of devices acting on their network.

The composition of the devices within the largest clusters in Figure 2 were then investigated to identify the common behavioural characteristics of each cluster. The characteristics of each device were resolved through dynamic host configuration protocol (DHCP) fingerprinting. DHCP fingerprinting allowed for the manufacturer and the operating system (OS) of the devices to be identified. Furthermore, detailed correspondence with the operators of the university's network provided additional insight into the function of managed devices on the network.

The investigation into the distinct clusters of devices in Figure 2 identified five main categories of device clusters: internal servers, public facing servers, user devices, lab computers, and printers. The following subsections provides additional information on each of the categories of clusters identified.

Servers – Two distinct server roles, internal **(1)** and public facing **(2)** were found via this clustering method. The servers on a network provide common resources and services which are often critical to the overall functionality of the network. Internal servers provide services to the devices within the network (e.g., a proxy server); whereas, public facing servers can provide services to both internal and external devices (e.g., a mail server). The identification of public servers on a network is of particular interest when assessing the vulnerability of the network as these devices are directly accessible from outside the network [6].

User devices (3) – User devices are unmanaged devices that are brought onto the network by staff and students to facilitate their work. The *high-activity* capture showed a significant increase in the number of user devices clustered on the network. This result is in correlation with the increase in staff and students present at the time of this capture. Furthermore, it was found that Apple devices in particular were easily identifiable via this method due to their frequency in connecting to Apple related services. Cluster 3.1 is highlighted in Figure 2 which contains 1,470 devices; of which, 90% could be confirmed to be an Apple device.

Lab computers (4) – Several computer labs are located on the university campus for which students can log onto the university network. Lab computers have a common base configuration due to the requirements of their respective lab. Therefore, lab computers were expected to form relatively well defined clusters. This insight appears to hold in the *low-activity* capture; however, the clusters of lab computers in the *high-activity* capture proved less distinct. This result was caused by the increase in use of these devices during the *high-activity* capture. Students not only use these devices for work related purposes but also for personal reasons (e.g., accessing social media). The varied use case in the *high-activity* capture is likely to have led to the varied behavioural patterns of these devices.

Printers (5) – Printers were found to form extremely well-defined clusters on the network. This result is due to the distinct network behaviour of printers. A printer is likely to connect to very few services via the internet. Typically, these services are register to the manufacturer of the printer itself. Therefore, a known printer that is not contained within a printer cluster would serve as a warning that the device is behaving uncharacteristically. Additionally, this warning indication could be applied to monitor the wide array of IoT related devices that would share similar network characteristics (E.g., IP cameras and smart sensors).

In summary, the observed device categories were found to be useful in acquiring initial insight into the composition of devices on a network. This insight could serve as a basis of analysis for network operators to identify the composition of devices operating on their network. Furthermore, this technique could serve as a tool for security analysts to identify potentially vulnerable device clusters on a network.

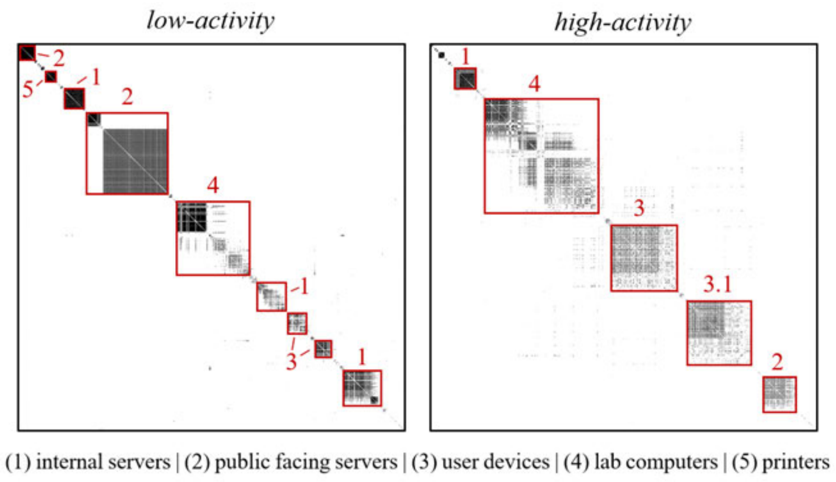


FIGURE 2. The ordered adjacency matrices of the two similarity graphs: *low-activity* (left) and *high-activity* (right). Five main categories of devices operating on the university's network were identified: internal servers (1), public facing servers (2), user devices (3), lab computers (4), and printers (5). Furthermore, a cluster of devices manufactured by Apple was also identified (3.1).

6. Conclusion and Future Work

In this paper, the application of affiliation graphs for clustering the devices on a large network was evaluated. It was found that the similarities between devices in an affiliation graph could be used to gain an initial insight into the composition of devices on a network. In particular, it was found that the devices on the university's network under analysis could be largely divided into five main categories; internal servers, public facing servers, user devices, lab computers, and printers. Additionally, it was found that Apple devices could be easily identified via this method due to the frequency in which they connected to Apple related services.

Future work will develop upon the methodology presented in this paper through the investigation of more sophisticated graph clustering techniques. In particular, graph embeddings will be used to cluster the devices on a network through the direct analysis of the affiliation graph. This extension is expected to improve the efficacy of the clusters generated whilst also removing the need to construct the similarity graph.

Acknowledgements

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

Appendix

Algorithm 1 Stochastic Local Clustering (SLC)

```
1:  set $\gamma > 0; n =$ number of iterations.
2:  for vertex $v$ in $A$ do
3:      $i = 0.$
4:      Set initial cluster, $D = \Gamma(v) \cup \{v\}$.
5:      Compute cluster performance, $f(D)$.
6:      while $i < n$ do
7:          Select a vertex, $u \in [D \cup \Gamma(D)], (u \neq v)$.
8:          If $u \in D, D' = D \setminus \{u\}$; else, $D' = D \cup \{u\}$.
9:          Compute new cluster performance, $f(D')$.
10:         set $D = D'$ if $X \sim U([0,1]) \leq e^{-\dfrac{f(D')-f(D)}{\gamma/log(i+2)}}$
11:         $i = i + 1.$
12:     end while
13: end for
```

where $\Gamma(v)$ is the neighbourhood of vertex, v; and γ is the starting temperature used in simulated annealing.

Simulated annealing (*step 10*) was used to ensure that the local search method did not get stuck in a local maxima. The cooling schedule used for simulated annealing was derived in [7]. A starting temperature, $\gamma = 0.2$, and number of iterations,

$n = 100$, was selected heuristically based on initial experimentation.

The metric of cluster performance, $f(D)$, as shown in (2) was derived in [14].

$$f(D) = \frac{2deg_{int}(D)^2}{|D|(|D|-1)(deg_{int}(D) + deg_{ext}(D)} \quad (2)$$

where deg_{int} and deg_{ext} is the internal and external degree of cluster D, respectively.

References

[1] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[2] Deepali Arora, Kin Fun Li, and Alex Loffler. Big data analytics for classification of network enabled devices. In *Advanced Information Networking and Applications Workshops (WAINA)*, pages 708–713.

[3] Stephen Borgatti and Daniel Halgin. Analyzing affiliation networks. *The Sage Handbook of Social Network Analysis*, 2011.

[4] Ronald L Breiger. The duality of persons and groups. *Social Forces*, pages 181–190, 1974.

[5] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. OS fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 173–180. ACM.

[6] Jeremy Faircloth. Penetration tester's open source toolkit. *Syngress*, page 32, 2016.

[7] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of operations research*, pages 311–329, 1988.

[8] Minzhao Lyu, Hassan Habibi Gharakheili, Craig Russell, and Vijay Sivaraman. Mapping an enterprise network by analyzing DNS traffic. In *International Conference on Passive and Active Network Measurement*, pages 137–152. Springer, 2019.

[9] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized IoT devices using machine learning techniques. *arXiv preprint arXiv:1709.04647*, 2017.

[10] Jeremy Merrill. Liberal, moderate or conservative? see how Facebook labels you. https://www.nytimes.com/2016/08/24/us/politics/facebook-ads-politics.html, 2016. [Online] Accessed on: September 2019.

[11] Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim. Operating system classification: A minimalist approach. In *ICMLC20 - International Conference on Machine Learning and Cybernetics - pending publication*.

[12] Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim. Characterising network-connected devices using affiliation graphs. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2020.

[13] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval*. McGraw Hill Book Co., New York, 1983.

[14] Satu Elisa Schaeffer. Stochastic local clustering for massive graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 354–360. Springer.

[15] Bruce Schneier. The internet of things will upend our industry. *IEEE Security & Privacy*, pages 108–108, 2017.

[16] John Skvoretz and Katherine Faust. Logit models for affiliation networks. *Sociological Methodology*, pages 253–280, 1999.

[17] Xuetao Wei, Nicholas C Valler, Harsha V Madhyastha, Iulian Neamtiu, and Michalis Faloutsos. Characterizing the behavior of handheld devices and its implications. *Computer Networks*, 2017.

[18] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *2012 IEEE 12th International Conference on Data Mining*, pages 1170–1175. IEEE.

[19] Guangxiang Zeng, Ping Luo, Enhong Chen, and Min Wang. From social user activities to people affiliation. In *2013 IEEE 13th International Conference on Data Mining*, pages 1277–1282. IEEE.

Statement of Authorship

Title of Paper	Detecting Data Exfiltration using Seeds based Graph Clustering
Publication Status	Published [✓ Accepted for Publication] Submitted for Publication Unpublished and Unsubmitted work written in manuscript style
Publication Details	Accepted in IEEE Asia-Pacific Conference on Computer Science and Data Engineering 2022

Principal Author

Name of Principal Author	Adriel Cheng		
Contribution to the Paper	Design, analysis, and interpretation of data. Drafted the manuscript.		
Overall percentage (%)	50%		
Signature		Date	27/9/2022

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception of technique. Assisted in design, analysis, and interpretation of data. Co-drafted the manuscript.		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	27 Sep 2022

Detecting Data Exfiltratation using Seeds based Graph Clustering

Adriel Cheng
Cyber and Electronic Warfare Division
Defence Science and Technology Group,
Department of Defence, Australia
adriel.cheng@defence.gov.au

Kyle Millar
Cyber and Electronic Warfare Division
Defence Science and Technology Group,
Department of Defence, Australia
kyle millar1@defence.gov.au

Identifying devices of interest within Internet Protocol (IP) networks is essential for Cyber security and network management. However, the large variety and number of devices within networks, and the ubiquitous encryption of network traffic poses significant barriers. We present a scalable and encryption resilient technique to identify devices of interest based solely on their affiliation to public internet services. In this work, devices of interest are those that are suspected of engaging with suspicious or unexpected servers; e.g., facilitating the malicious exfiltration of data. Graph-based clustering was used to reveal devices of interest based on the similarity of their network behaviour to a set of pre-known malicious devices acting as seeds. The motivation for our technique was driven by a case study for which a subset of malicious devices were known; however, the vast majority remained undiscovered. We conducted experiments to demonstrate viability of our technique as applied to this use-case. Detection accuracies of 94% were achieved, and malicious devices from the case study were successfully identified to aid data driven decisions by Cyber analysts.

Keywords - Cyber Security, Network Security, Graph Clustering, Machine Learning

I. INTRODUCTION

Identifying devices of interest within Internet Protocol (IP) networks is essential for both Cyber security and network management. When networks are infiltrated by malicious command-control (C2) activity, identifying victims or perpetrators of such activity is increasingly difficult. This is due to the growth of networks in both the number and types of devices and their traffic volumes and diversity. Furthermore, the traffic flowing amongst devices are now predominately encrypted [1,2], making it impossible to examine traffic content to find devices of interest for different applications.

In this paper, we present a technique to identify anomalous network devices (i.e. their IP address) by their network graph topological information. The network graphs are extracted from network traffic flows metadata only, overcoming network traffic encryption barriers and maintaining user privacy.

Whilst the need for detecting anomalous devices are diverse, we focus on network monitoring and analysis in the Cyber security domain. In particular, we target the use-case of uncovering malicious devices operating on a network.

A. *Application Use-Case*

Our technique was inspired by the application use-case depicted in Fig. 1. The use-case involved detecting devices uploading excessive amounts of data to unexpected or suspicious internet servers. Such devices are suspected to be victims of Cyber C2 malicious attacks; and have been compromised such that data is being exfiltrated to external servers controlled by the attacker.

A practical scenario involves administrators of an organization being presented with evidence of stolen confidential data (e.g. competing companies duplicating technology, or via investigative or intelligence reports); and they seek to identify which device(s) on their networks is leaking this sensitive data [1]. Beyond the Cyber C2 example, another malicious example is insider threat scenarios where disgruntled staff transfer information externally without permission.

Previously, in order to identify such compromised devices (or users), a manually intensive process was required. Considerable time and effort was required by analysts to examine the network traffic of devices; e.g. to identify and correlate devices sending unusually large volumes of traffic to external servers. Whilst such excessive and suspicious network activity are detectable if sufficient access to devices [2] and internal networks are afforded, when limited to network gateway monitoring points (i.e. at the edge of enterprise networks or ISPs), manually driven procedures are no longer viable given encryption and the scale of networks and traffic flows.

B. *Overview of Technique*

Our technique relies on observing the traffic flow connections between network devices and the internet. Specifically, we are interested in the connection profiles and associated network traffic of the devices – i.e. what does a device connect to and when. The technique detects if certain devices are connected to unexpected internet servers whilst exhibiting unusual traffic uploading activity. Devices exhibiting such characteristics may indicate Cyber exfiltration of data.

Beyond data exfiltration, other uses-cases involve detecting devices that utilize services or sites that promote illegal or undesirable activities. For instance, in most organizations, one would not expect devices (i.e. users) to connect to online gaming or gambling sites as part of normal business functions. Whilst it

[1] Data leaks may originate from various sources, but for this paper we are interested in data exfiltration from internal network devices.

[2] Note the growth of bring-your-own devices beyond the control of network admins has grown substantially in certain organizations [3], e.g. universities.

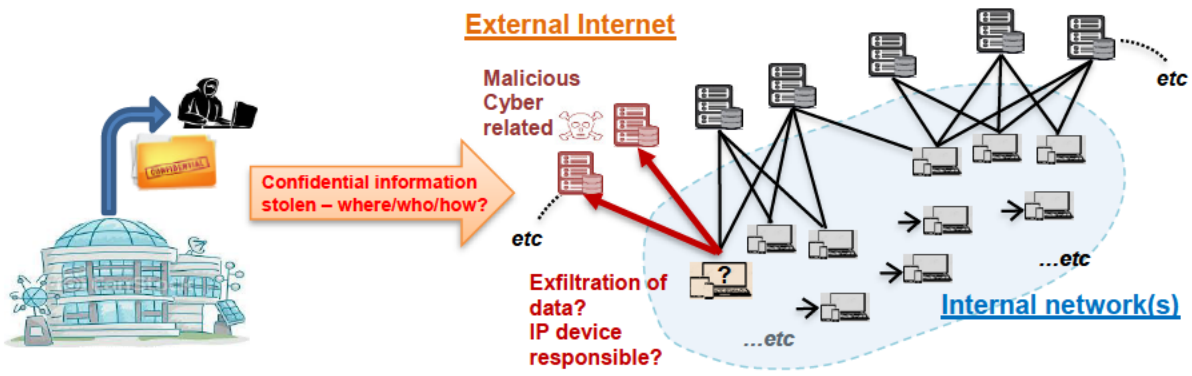

External Internet
Malicious Cyber related
Confidential information stolen – where/who/how?
etc
etc
Exfiltration of data? IP device responsible?
...etc
...etc
Internal network(s)

Fig. 1. Application use-case scenario – detect and identify data exfiltration from device(s)

is trivial to blacklist pre-known sites or malicious servers, this is not sustainable for many other unknown prohibited sites, or when new forbidden services/sites are created in the future.

Instead, an alternative approach is to observe existing devices connecting to pre-known servers/sites of interest. We examine the traffic behaviors and connection profiles to illicit sites, learn these profiles and identify other similar devices associated with such sites operating currently or newly created in the future.

Our technique creates *affiliation* graphs of internal network devices to external servers across the internet. Affiliation graphs capture the network connection profiles of devices. Fig. 2 shows the conceptual overview of our method. We first ascertain which internet servers are relevant to our application use case (e.g. C2 servers receiving stolen data or sites facilitating criminal activities). Next, we examine for anomalous behaviors associated with these servers (e.g. overly large data uploads to C2 related servers).

To identify these servers, we make use of prior known devices of interest, termed *seeds*, whose connection profile and network traffic behaviors to external servers indicate anomalies. We denote groups of these servers as communities of interest, which represent groups of individual server IP addresses or IP subnets in the internet. Using these communities of interest, we build and examine the affiliation graphs to find other devices that exhibit similar network traffic behavior and affiliate with similar sets of communities as the seed devices.

Specifically, graph clustering methods are applied to identify other IP devices similar to the seeds. We seek to detect new devices connecting to similar communities that also exhibit the same traffic flow characteristics as the seed devices. Furthermore, unknown communities of interest could be uncovered through analyzing the connection profile of newly discovered devices of interest.

C. Paper Contributions and Summary

The proposed technique extends our work previously in [4,5,6] whereby affiliation graphs were created to infer the operating systems of devices. In this paper, we extend and apply seeds based graph clustering methods to affiliation graphs data.

Our contributions in this paper are as follows:

- We devise a seeds based network device detection technique using affiliation graph structures that is resilient to encryption barriers.
- We present a real-life use-case to demonstrate the efficacy of the technique. Through our application use-case, we demonstrate our technique uncovers anomalous devices to aid analysts in identifying devices involved in data exfiltration related activities.
- Finally, we provide insight into the selection of seeds for detecting malicious devices.

The rest of this paper is as follows. Section II provides preliminary definitions for the seeds based graph clustering method described in Section III. Experiments and results are presented and discussed in Sections IV and V. We survey related work in Section VI, before the paper concludes in Section VII.

II. Preliminaries – Affiliation and Similarity Graphs

We define an affiliation graph [4] as a bipartite graph consisting of two distinct sets of vertices, actors (A) and communities (C) (see Fig. 3). An actor ($a \in A$) is affiliated with a community ($c \in C$) if there exists a membership ($m_{a,c} \in M$) between a and c, where M is the set of all memberships between A and C. An affiliation graph is represented as the collection of actor, community, and membership sets, $G = (A, C, M)$.

For the use-cases described in this paper, actors are IP addresses belonging to network devices; and communities are IP addresses (or subnets) of various internet sites or servers that the actors interact with. Relying simply on these two IP address fields, our technique is immune to barriers imposed by encrypted network traffic that limit the information available for analysis. The availability of such address fields also ensure our technique can be easily applied to other types of networks with different observation points. Our affiliation graphs are constructed [4] such that the weight of a membership $\omega(m)$, is the number of flows between a device and a community.

Building upon the affiliation graph, a *similarity* graph links actors (i.e. devices) that share strong similarities with other actors according to the overlap in their affiliated internet communities. A similarity graph $G_A = (A, E)$ can be constructed for which the edges E is the set of all similarities between actors in A of the affiliation graph G.

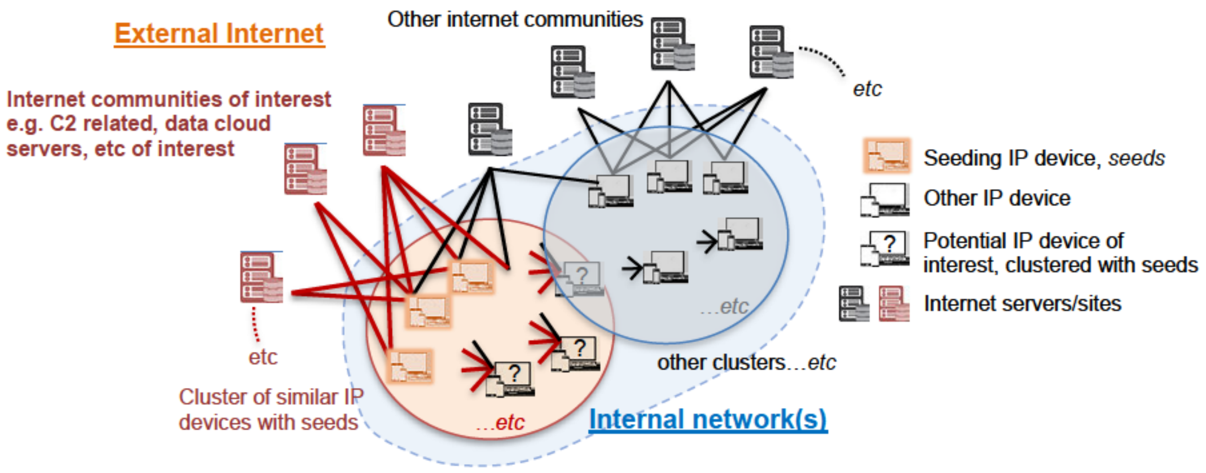

External Internet
Other internet communities
etc
Internet communities of interest e.g. C2 related, data cloud servers, etc of interest
Seeding IP device, seeds
Other IP device
Potential IP device of interest, clustered with seeds
Internet servers/sites
...etc
etc
other clusters...etc
Cluster of similar IP devices with seeds
...etc
Internal network(s)

Fig. 2. Seeds based affiliation graph clustering concept for detection of IP devices

The weight of the edges in E signify the level of similarity between actors. We use Salton's cosine coefficient [8] to quantify edge weights. Salton's cosine coefficient takes into consideration the frequency in which two actors connect to the same community. A community which is frequented more often signifies a greater dependence that the device has on the community. Two devices sharing a strong dependency are likely to be more similar than two devices which share a once off connection.

The similarity between two actors, a_1 and a_2, as computed by the Salton's cosine coefficient of their membership sets is $sim(a_1, a_2) = \frac{M(a_1).M(a_2)}{\|M(a_1)\| \times \|M(a_2)\|}$ where $M(x)$ is the membership vector of an actor x, $\|M(x)\|$ is the length of vector $M(x)$, and $M(x) = \{\omega(m_{x,c}): c \epsilon C\}$. If $m_{x,c} \notin M$ then $\omega(m_{x,c}) = 0$.

In Fig. 3, we depict an example similarity graph extracted from an affiliation graph. Fig. 3 shows that a_2 and a_3 share the strongest similarity arising from the number of (weighted) flow connections to the same three communities c_1, c_2 and c_3 in the affiliation graph. Correspondingly in the similarity graph, a_2 and a_3 share a larger weighted (thicker) edge to denote this similarity. In contrast, the remaining edges are less similar due to the lower number of commonly shared destination communities (i.e. only two communities); and are indicated by the thinner edges in the similarity graph.

Finally, using similarity graphs, Stochastic Local Clustering (SLC) [15] is applied to automatically group actors that share similar connection profiles to their internet communities. SLC (see Algorithm 1) performs graph traversals specifically for forming clusters from graph datasets. We use SLC because it does not require the entire graph to be processed in order to acquire clusters [7]; unlike other clustering methods whereby the entire graph must be analysed beforehand.

III. SEEDS BASED GRAPH CLUSTERING

Armed with affiliation and similarity graphs of network devices, we present the seeds based graph clustering technique. Fig. 4 shows the overall flow of the method. First, network traffic flows from the internet in (A) are used to build an affiliation graph. In our affiliation graphs (B), the actors are the IP addresses of network devices within an internal network such as an enterprise network or ISP. The communities are the internet servers or sites that the actors connect to. The affiliation graphs encode the connection profiles of the actor devices. Like previous work [4], it is from these connection profiles that we seek to infer behaviours and device information about the actors. Next in (C), similarity graphs are extracted from affiliation graphs. A similarity graph links actors which are considered similar to each other in terms of their internet connection profiles.

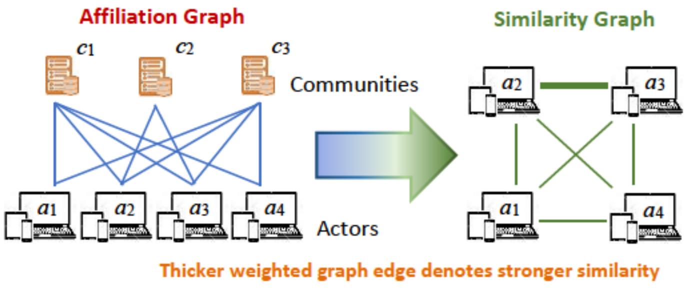


Fig. 3. Affiliation to similarity graph extraction

```
1: Set γ > 0; n = number of iterations
2: for vertex v in A do
3:     i = 0
4:     Set initial cluster, D = Γ(v) ∪ {v}
5:     Compute cluster performance, f(D)
6:     while i < n do
7:         Select a vertex u ∈ [D ∪ Γ(D)], (u ≠ v)
8:         if u ∈ D, D' = D \ {u}; else D' = D ∪ {u}
9:         Compute new cluster performance, f(D')
10:        if X ~ U([0,1]) ≤ $exp\left(-\frac{f(D')-f(D)}{\gamma/\log(i+2)}\right)$, set D = D'
11:        i = i + 1
12:    end
13: end
```

$\Gamma(v)$ is defined as the set of all vertices with an edge connecting to v. Simulated annealing (step 10) was used to ensure the local search method is not stuck in a local maxima. A starting temperature of $\gamma = 0.2$ was selected. The cluster performance metric $f(D)$ was derived in [15], $f(D) = \frac{2deg_{int}(D)^2}{|D|(|D|-1)(deg_{int}(D)+deg_{ext}(D))}$ where deg_{int} and deg_{ext} are the internal and external degree of cluster D respectively.

Algorithm 1. Stochastic local clustering algorithm

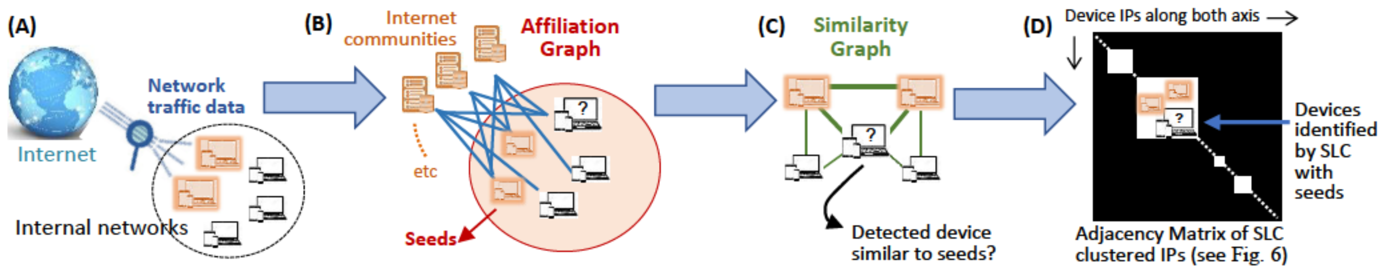

(A)
Internet
Network traffic data
Internal networks
(B)
Internet communities
Affiliation Graph
etc
Seeds
(C)
Similarity Graph
Detected device similar to seeds?
(D)
Device IPs along both axis
Devices identified by SLC with seeds
Adjacency Matrix of SLC clustered IPs (see Fig. 6)

Fig. 4. Seeds based graph clustering steps

Using the similarity graph, SLC is then applied to cluster and identify groups of IP devices that share a high degree of similarity in their connection profiles.

In order to find actor devices of interest, in (D), we make use of pre-identified devices, termed *seeds*. Seeds are prior known anomalous devices of interest that are used to identify other unknown devices of significance. Our technique aims to find clusters produced by SLC containing such seeds; and we infer other devices clustered together with seeds are also anomalous devices of interest as they affiliate with similar internet communities and share similar characteristics as our seeds. We visualize our clustered results using adjacency matrices (see Sec. V.A).

Our hypothesis relies on the number of seeds within a cluster. The greater the number of seeds the more likely other devices within the cluster are similar to the seeds; and are considered of interest to the use-case application. In contrast, clusters that do not contain a seed identify devices that are likely not of interest.

This approach provides dual benefits: (i) potential devices of interest are automatically identified in clusters, and (ii) all other irrelevant devices in non-seeded clusters can be discarded. Hence, even if some devices within seeded clusters did not turn out to be of interest to the application use-case, the majority of irrelevant devices would still be filtered away leaving behind a much smaller set of device IPs to manually investigate.

This approach is particularly useful if limited number of seeds are available. With a lower number of seeds, the confidence in accurately identifying unknown devices of interest is lower. However, a large percentage of irrelevant devices from non-seeded clusters would still be filtered away, reducing the number of devices to examine.

Formally, we describe our seeds based clustering and device detection method as follows. First, we find clusters containing seeds. Let Ω' be the set of clusters containing at least one seed from the set of seeds, S. That is, $\Omega' = \{D \in \Omega \mid |D \cap S| > 0\}$ where Ω is the set of all clusters acquired from the similarity graph using SLC in Algorithm 1, $D \in \Omega$ is an arbitrary cluster of IP devices from Ω.

Using Ω', we find unknown devices of interest. Let I be the set of devices of interest uncovered by the seeds based graph clustering method, such that $I = I \cup D' \setminus S, \forall D' \in \Omega'$ where D' is a cluster containing at least one seed. Hence, the set I contains the device IPs that are not seeds but are clustered with seeds from S.

IV. Experimentation

Our experimental setup was depicted in Fig. 4(A). We monitor traffic in and out of an internal network to the external internet. The internal network is typically an enterprise network, though the network could also be an amalgamation of multiple networks, or much larger sized networks such as the class A or B subnets used by tier 1 ISPs or very large organizations. The internal network contains the actor IP devices, and the external network traffic between these devices and the internet community IPs are captured for analysis.

We apply our technique to captured traffic transiting in and out of the network at an observation point such as an internet gateway. In our set up, over a 7 day period, the internal network contained tens of thousands of devices; and externally, there may be millions of internet community IPs. The community IPs are aggregated into IP subnets, hence we deal with much smaller number of communities. Overall, there were 76 million flows representing 4.6TB of traffic transiting the observation point.

A. *Datasets, seeds, and experimental metrics*

TABLE I shows the 3 sets of data used for our experiments. We aim for a variety of network samples to test the robustness of our technique. For each dataset, we create affiliation and similarity graphs resulting in graph sizes and the number of actor and community subnet IPs stated. A subset of the actor IPs are designated as seeds – i.e. pre-known malicious devices. The set of seeds S are identified manually based on their network traffic and community connection profiles. Specifically these seeds are involved in large traffic data transactions to pre-selected community IPs, and are of interest to our application use-case as outlined in Sec. I.A.

For experimental purposes, we allocate seeds into training and testing sets S_{tr} and S_{ts} respectively, where $S = S_{tr} \cup S_{ts}$. Unlike conventional supervised machine-learning, we denote training and testing seeds as follows. Training seeds S_{tr} are used to identify device clusters of interest. S_{tr} represents pre-known devices whose characteristics epitomise the behaviours of interest we seek to detect in other devices. Once clusters containing seeds from S_{tr} are identified, we examine these clusters to evaluate how many devices in these clusters belong to the test set S_{ts}. The goal is to identify as many devices from S_{ts} as possible within seeded clusters containing S_{tr}.

TABLE I. Dataset samples

Dataset	# flows	# actors	# communities	# edges	# seeds S
1	28,172	13,788	24	27,986	3,000
2	61,898	32,831	9,980	51,666	3,000
3	53,220	12,271	17,352	34,707	3,000

TABLE II. Precision, recall, F1-scores

	Dataset	Precision	Recall	F1-score
Training seeds $\lvert S_{tr}\rvert$ = 500	1	0.99	0.74	0.85
	2	0.99	0.67	0.80
	3	0.99	0.68	0.81
Training seeds $\lvert S_{tr}\rvert$ = 1000	1	0.99	0.88	0.94
	2	0.99	0.86	0.93
	3	0.99	0.72	0.84

For seeds based graph clustering, we define the precision and recall metrics as follows. For every cluster with at least one training seed from S_{tr}, precision measures the number of devices in the cluster belonging to the test set S_{ts} compared to the total number of devices in that cluster. Precision queries the following: *Out of all the devices suspected to be of interest in a cluster, how many devices were accurately identified as malicious.*

Recall measures how many devices from the test set S_{ts} are clustered within a cluster containing at least one training seed from S_{tr}. Recall addresses the following: *Out of all the devices in test set S_{ts}, recall measures how many malicious devices were detected by our technique*. And finally, F1-score is computed from the harmonic mean of precision and recall.

For our similarity graphs, note that a minimum similarity threshold is enforced to ensure only source actor IPs that are sufficiently similar to one another are included. We use Salton's cosine coefficient from Sec. II as the measure of similarity, and an empirically measured threshold τ of 0.8 similarity is used; such that actor IPs a_1 and a_2 with similarity lower than $\tau = 0.8$ are excluded from the clustering phase.

V. Results

We first assess our technique for uncovering anomalous devices of interest similar to our seeds. TABLE II shows the precision, recall and F1-scores using 500 and 1000 training seeds. The results indicate near 100% precision across all datasets. The high precision shows that a device is highly likely to be of interest if clustered with a seed device. Furthermore, high precision also indicates that the majority of devices from the seeds test set were detected. This confirms the seeded clusters produced by our technique uncovered other devices that exhibit large data upload behaviours to malicious exfiltration servers; as desired by our application use-case.

In terms of recall, 0.67 to 0.88 recall was attained. High recall is difficult to achieve because identifying anomalous devices or finding clusters containing such devices depends on the availability of *useful* seeds – i.e. the number of such seeds and how *influential* these seeds are. Our seeds must provide distinct patterns of network flow connections, including extremely large volumes (bytes) of network traffic. As greater number of influential seeds available for use are applied, higher recall is achieved (see Fig. 5). Overall, the F1-socres in TABLE II confirms viability of our technique. Generally, with at least 500 training seeds, F1-scores of up to 0.8 is considered favourable.

Fig. 5 shows the increase in precision, recall and F1-scores as the number of training seeds are added. This result is as expected; i.e. additional training seeds generally produces a greater number of seed based clusters to identify other anomalous devices of interest from the seeds test set.

Generally, the selection of seeds affect how IP devices of interest are clustered and detected. In particular, seeds which exhibit distinct characteristics different to benign devices are beneficial for our technique. To demonstrate this, we compared the selection and use of training seeds *ordered* by volume of bytes transacted versus the number of flows (i.e. aggregated connections between network endpoints).

For our application use-case, the goal was to detect devices involved in excessive data uploads to malicious internet communities. Using 500 seeds ordered by number of flows, we achieved between 0.65 to 0.74 F1-scores. The results are lower than those in TABLE II, which used seeds ordered by bytes. Intuitively, this is to be expected. Excessive traffic volumes exhibited by devices involved in data exfiltration are more easily differentiated by number of bytes instead of number of flows when compared to other devices. Hence, when deploying our technique, consideration must be given to how seeds are chosen and ranked for use.

A. Cluster Visualization

Next, we visualise our seeded clusters in Fig. 6. We show an adjacency matrix of Salton Cosine similarity values between each pair of devices in the dataset. The devices along both axis of the matrix are re-ordered such that devices within the same clusters are grouped together highlighting prominent clusters. White or lighter shaded pixels within the matrix represent strong similarities between devices, whilst darker pixels represent low or nil similarity.

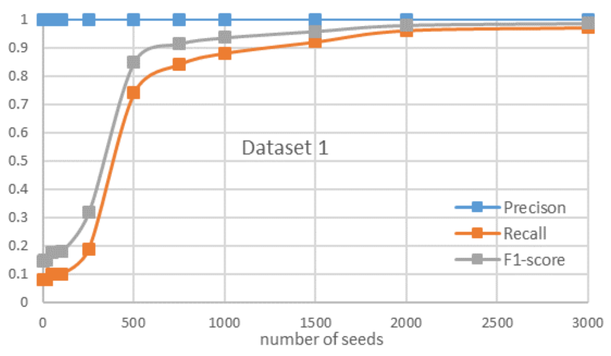
The adjacency matrix shows distinct clusters of varying sizes are acquired by our technique. Significantly, the most prominent clusters in terms of similarity (i.e. most white/lighter blocks along diagonal) contain our training seeds. This is clearly shown in Fig. 6 (top) where seeds were ordered by bytes; in contrast to Fig. 6 (bottom) with seeds sorted by flows. As explained in the prior section, for our application use-case, using seeds ordered by bytes is more effective for identifying devices exhibiting excessively large traffic data uploads. Finally, if sufficient number of training seeds are used, then the majority of these clusters would also assist in identifying additional (testing) seeds of interest.

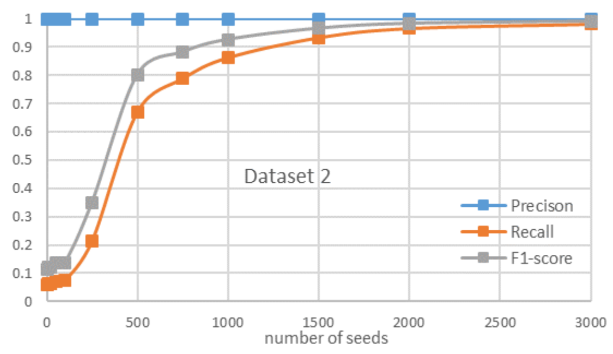
B. Application Use-Case Discussion

Finally, we discuss the results based on our application use-case in Section I.A. Recall we aim to identify the malicious device(s) involved in exfiltrating data to a set of malicious servers. To find such malicious device(s), we use training seeds that epitomise known malicious behaviour of interest. The adjacency matrix in Fig. 6 (top) showed that unknown malicious device(s) can indeed be uncovered by our technique.

For example, a particular malicious device, unbeknownst within the training seeds, was shown to be located within seeded clusters (94 and 74) of the captured data. Up to 500 training seeds were selected by our technique to acquire these clusters. These training seeds are the top 500 seeds when ordered by bytes – i.e. the top 500 seed devices uploading the largest volumes of traffic data (when measured by number of bytes) to suspicious community servers.

The cluster containing the malicious device includes at least one (or more) of these 500 seeds. Using a more refined selection





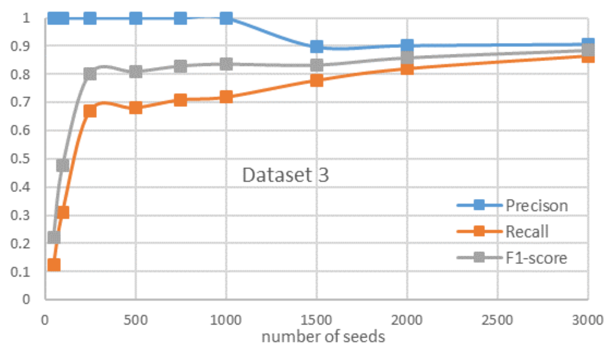
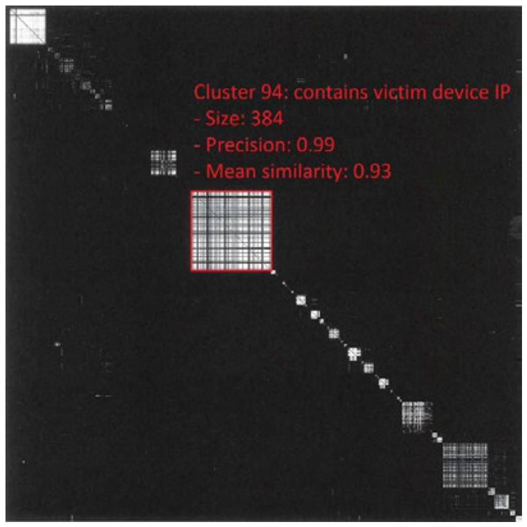


Fig. 5. Precision, recall and F1-scores versus number of seeds



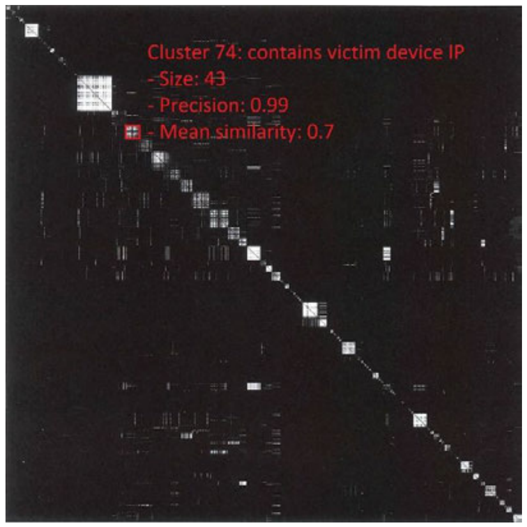


Fig. 6. Cluster visualization using adjacency matrices with devices listed along x and y axis; using Dataset 2 and with seeds ordered by bytes (top) and ordered by flows (bottom)

of seeds, we are confident a smaller cluster containing the malicious device would be provided. Regardless, by simply using our technique with these 500 seeds, the manual search space for finding the malicious device has reduced significantly from >30,000 original devices to only 384.

Whilst using 500 seeds may appear excessive and difficult to acquire, we observe it is possible to use seeds from different networks; similar to how commercial intrusion detection systems share signatures from multiple networks or devices. Alternatively, the seeds could also be synthesized, but we designate this as future work.

Finally, we note that the malicious device was included in a cluster with high precision, indicating all devices clustered together belonged to the seeds test set. Hence, in the reverse scenario, if the identified malicious device had been used as a seed, and the goal is to uncover other unknown but similar malicious devices, then this cluster would have revealed 383 other devices of interest to investigate. The degree of confidence that these 383 devices are anomalous is also supported by the high level of similarities between every device in the cluster. Specifically, a mean similarity of 0.93 measured by our Salton Cosine coefficient was attained for this cluster.

The technique described in this paper aims to assist analysts in the discovery of devices exhibiting specific network characteristics – i.e. various characteristics may be considered, but in our case, excessive data uploads to suspicious internet servers is our focus. In our experience, directly identifying IP addresses of malicious devices is difficult and requires further verification regardless. Hence, our technique presents a reduced set of potentially suspicious devices to analysts; to semi-automate and speed up the detection of malicious devices(s).

VI. Related Work

The use of affiliation graphs for clustering was inspired by our earlier work in device characterisation. The goal in [4,5,6] was to characterise the operating systems (OS) of mobile devices. These approaches were purely supervised. In contrast, the technique presented within this paper relies on seeding information of prior known devices to guide the anomalous device detection process.

Our technique for identifying IP devices relates most to network traffic anomaly detection using clustering methods. In [10], clusters of homogenous traffic examined over time represent normal baseline traffic. When new clusters of different sizes and types of traffic are found, such deviations from baselines signify traffic anomalies. In [11], anomaly detection is formulated as a graph clustering problem. Graph properties from network traffic flows are partitioned using their *NodeClustering* technique to separate normal from abnormal traffic nodes. Similarly in [12], a graph partitioning cluster method is utilised to detect anomalies. And finally in [13], network traffic is converted into first and second order graphs representing individual IP endpoints and the general network. Extracted features from these graphs are then supplied to machine learning methods for anomaly detection.

The methods surveyed above use statistical latent features and network graph embeddings from traffic flows data for clustering. In contrast, our technique only relies on the source and destination endpoints of bipartite graphs and the links between them. In this sense, our technique can be applied more generally as it relies only on minimal network flows data that is widely available. In other work [14], we extracted graph embeddings explicitly from our affiliation graphs. However, in this paper, our affiliation graphs are transformed into similarity graphs that reveal similarity links between nodes to cluster common IP devices. In this sense, our technique directly exploits the similarity of device behaviour to identify devices of interest.

The surveyed techniques [10-13] also focus on detecting disparities between normal and abnormal network characteristics. In comparison, our method aims to find similar anomalous IP nodes. We rely on the use of seed devices that are abnormal, in order to cluster and find other similar devices of interest. Whilst other techniques do not rely on such seeds, our approach is explicitly designed to uncover distinct anomalous characteristics amongst devices; hence we achieve higher likelihood of detection success.

Finally, our technique relies on Stochastic Local Clustering (SLC); which performs graph traversals specifically for forming clusters from graph datasets compared to other distance or fuzzy based clustering methods in [10-13]. SLC does not require the entire graph to be processed in order to acquire clusters [7,15]. Avoiding full graph analysis is highly beneficial from a scalability perspective given the large sizes of network traffic graphs we deal with.

VII. Conclusions and Future Work

Detecting and identifying anomalous devices of interest is extremely difficult given ubiquitous traffic encryption and the excessively large number of network devices to examine. In this paper, we presented a seeds based graph clustering technique to aid device detection by matching network behaviours of other similar but pre-known devices of interest.

Experiments based on real-life network traffic data from an example application use-case demonstrated the efficacy of our technique. We focused on identifying Cyber compromised malicious devices involved in exfiltration of data to C2 servers. Our results showed that previously undetected malicious devices were detected with high precision.

Our technique is most effective at reducing the scope and size of the IP devices search space; to provide a reduced set of devices that are more likely to be of interest to the target use-case application. In developing this technique, our intention is to aid automation of IP device detection and reduce manual efforts of Cyber analysts where possible.

In the future, we hope to incorporate the addition of other traffic flow attributes into the network graphs for clustering. For example, the inclusion of flow directionality into the graph edge weights. Another avenue of investigation is an iterative cluster approach inspired by [9] to continually refine production of new clusters and further enhance detection of anomalous devices.

References

[1] Google, "Google Transparency Report: HTTPS encryption on the web," 2022. [online accessed 26-4-2022]. Available: https://transparencyreport.google.com/https/overview?hl=en%7D%7D..

[2] G. Aceto, G. Bovenzi, D. Ciuonzo, A. Montieri, V. Persico and A. and Pescapé, "Characterization and Prediction of Mobile-App Traffic Using Markov Modeling," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 907-925, March 2021

[3] X. Wei, N. Valler, H. Madhyastha, I. Neamtiu and M. Faloutsos, "Characterizing the Behavior of Handheld Devices and its Implications," Computer Networks, vol. 114, pp. 1-12, 2017.

[4] K. Millar, A. Cheng, H. Chew and C. Lim, "Characterising Network-Connected Devices Using Affiliation Graphs," in GraSec (Graph-based network security) workshop at the IEEE/IFIP Network Operation and management symposium (NOMS), pp. 1-6, 2020.

[5] K. Millar, A. Cheng, H. Chew and C. Lim, "Operating System Classification: A Minimalist Approach," in IEEE International Conference on Machine Learning and Cybernetics (ICMLC), pp. 143-150, 2020.

[6] K. Millar, A. Cheng, H. Chew and C. Lim, "Clustering Network-Connected Devices using Affiliation Graphs," in IEEE International Conference on Machine Learning and Cybernetics (ICMLC), 2021.

[7] S. E. Schaeffer, "Graph Clustering - Survey," Computer Science Review Elsevier, pp. 27-64, 2007.

[8] G. Salton and M. McGill, Introduction to Modern Information Retrieval, New York: McGraw Hill Book Co., 1983.

[9] J. Davis, A. Cheng, J. Hefferan, L. Singh and D. Webb, "Iterative Clustering and Filtering for Guided Cyber Discovery," DST-GROUP-TR-3647, 2018.

[10] G. Lieto, F. Orsini and G. Pagano, "Cluster Analysis for Anomaly Detection," in Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08, 2009.

[11] H. Alene, K. Hatonen and P. Halonen, "Graph Based Clustering for Anomaly Detection in IP Networks," 2011.

[12] M. Ahmed and A. Mahmood, "Novel Approach for Network Traffic Pattern Analysis using Clustering based Collective Anomaly Detection," Annals of Data Science, vol. 2, pp. 111-130, 2015.

[13] Q. Xiao, J. Liu, Q. Wang, Z. Jiang, X. Wang and Y. Yao, "Towards Network Anomaly Detection Using Graph Embedding," in International Conference on Computational Science, Lecture Notes in Computer Science, pp. 156-169, 2020.

[14] K. Millar, A. Cheng, H. Chew and C. Lim, "Enhancing Situational Awareness in Encrypted Networks using Graph-based Machine Learning," IEEE Transactions on Network and Service management, vol. Under Submission, 2022.

[15] S. E. Schaeffer, "Stochastic Local Clustering for Massive Graphs," in Pacific-Asia Conference on Knowledge Discovery and Data Mining, Lecture Notes in Computer Science, pp. 354-360, 2005.

Statement of Authorship

Title of Paper	Operating System Classification: A Minimalist Approach
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Operating System Classification: A Minimalist Approach," in International Conference on Machine Learning and Cybernetics (ICMLC), 2020: IEEE, 2020, pp. 143-150.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Operating System Classification: A Minimalist Approach

Kyle Millar*, Adriel Cheng*,†, Hong Gunn Chew*, and Cheng-Chew Lim*
*School of Electrical and Electronic Engineering, The University of Adelaide, Australia
†Cyber and Electronic Warfare Division, Defence Science & Technology Group, Australia
Email: {kyle.millar, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au*
Email: adriel.cheng@dst.defence.gov.au†

***Abstract*—Operating system (OS) classification is of growing importance to network administrators and cybersecurity analysts alike. The composition of OSs on a network allows for a better quality of device management to be achieved. Additionally, it can be used to identify devices that pose a security risk to the network. However, the sheer number and diversity of OSs that comprise modern networks have vastly increased this management complexity. We leverage insights from social networking theory to provide an encryption-invariant OS classification technique that is quick to train and widely deployable on various network configurations. In particular, we show how an affiliation graph can be used as an input to a machine learning classifier to predict the OS of a device using only the IP addresses for which the device communicates with.**

We examine the effectiveness of our approach through an empirical analysis of 498 devices on a university campus' wireless network. In particular, we show our methodology can classify different OS families (i.e., Apple, Windows, and Android OSs) with an accuracy of 99.3%. Furthermore, we extend this study by: 1) examining distinct OSs (e.g., iOS, OS X, and Windows 10); 2) investigating the interval of time required to make an accurate prediction; and, 3) determining the effectiveness of our approach after six months.

***Index Terms*—Cybersecurity, machine learning, affiliation graphs, operating system classification, passive network reconnaissance.**

I. Introduction

Operating system (OS) classification aims to identify the OS of a device (e.g., iOS, Android, or Windows 10) solely through the observation of its network traffic [2]. Identifying the composition of OSs on a network allows a network administrator to provide a better quality of service (QoS) and assists cybersecurity analysts in identifying vulnerable devices [4]. However, with the growing number and diversity of OSs, implementing a comprehensive OS classification technique is becoming increasingly difficult in practice.

Current OS classification techniques have typically relied on either a large number of difficult to extract features (e.g., Internet protocol (IP) ID monotonicity and clock frequency) or target a specific subset of OSs [7]. The wide array of devices present on modern networks renders these techniques computationally ineffective at scale. Additionally, the wide spread adoption of encryption standards has reduced the ability to extract these rich feature sets from packet data. It has therefore become necessary for OS classification techniques to rely on as little information as possible.

The aim of this paper is to produce an OS classifier that uses only the bare minimum of information that would be available to a network administrator. To achieve this aim, we propose the use of **affiliation graphs** to construct a behavioural profile of a device by examining the IP addresses that the device communicates with via the Internet. In restricting our analysis to just these IP addresses, we create an encryption-invariant analysis that is quick to populate and requires little storage. An example of constructing an affiliation graph from IP addresses is shown in Figure 1.

Affiliation graphs are a social networking technique that investigates the characteristics of an individual (denoted as an *actor*) through the communities they are affiliated with [6]. Social communities form through both implicit and explicit

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

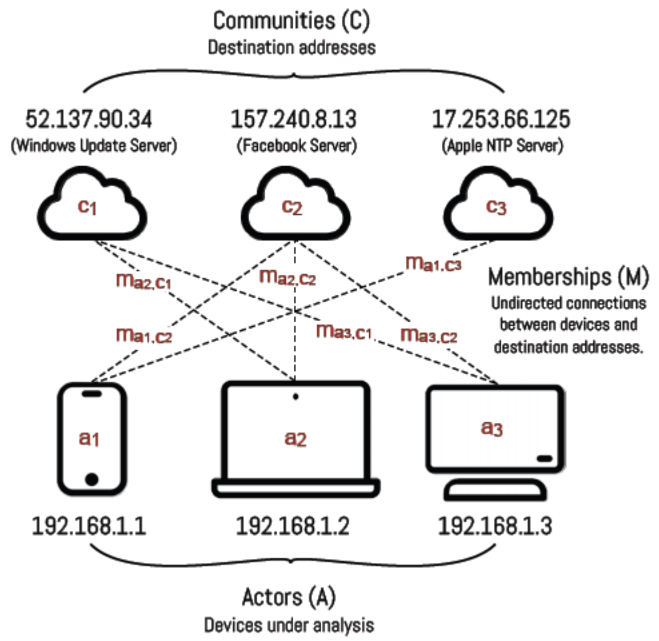


Fig. 1. An affiliation graph created from network data. The actors and communities of the affiliation graph are the devices internal and external to the network under analysis, respectively. The membership set consists of the network communications between the internal and external devices.

commonalities in their membership set; such as shared interests, occupation, or genealogy [14]. Through analysing the shared commonality within a community, certain characteristics of its membership set can often be revealed. For example, Facebook is able to identify the political alignment of a user based solely on the Facebook groups the user is affiliated with [11].

The advantage of an affiliation graph lies in the ease in which the affiliations between an actor and their communities can be observed [5]. Through understanding why these affiliations are made, insightful inferences can be made about an actor without requiring any *a priori* knowledge of the actor themselves.

It is the axiom of this work that the OS of a device (*actor*) can be determined by the IP addresses (*communities*) that the device affiliates with via the Internet. For example, a device affiliated with a Windows update server is likely to be running the Windows OS [7]. A generalisable behavioural profile of distinct OSs could therefore be constructed given the knowledge of the communities that an OS will natively affiliate with.

The key contribution of our work is threefold:

1) We show that a random forest (RF) classifier trained on an affiliation graph can achieve a 99.3% classification accuracy for distinct OS families (i.e., Apple, Windows, and Android OSs). Furthermore, a 94.6% classification accuracy could also be achieved for distinct OSs (e.g., iOS, OS X, and Windows 10).
2) We show that it takes approximately one hour to construct a representative profile of an OS based solely on the community IP addresses it communicates with via the Internet. Monitoring an OS for longer than one hour results in minimal improvement to the overall OS classification performance.
3) We examine the deterioration of the classification performance of this methodology after six months. We show that there is a relatively small decrease in overall accuracy for both Apple and Android OSs (3% and 7%, respectively); however, a significant decrease of 29% was observed for Windows OSs. This deterioration can be mitigated by retraining the classifier every few months.

The remainder of this paper is structured as follows: A brief survey of related work in OS classification is given in Section II. Section III introduces affiliation graphs and provides an overview of the data set used for this analysis. Section IV introduces a feature selection process to select the relevant communities for OS classification. Results for OS Family and OS classification are given in Section V. Section VI investigates the amount of time required to make an accurate prediction of the host device's OS. Section VII examines the classification performance of the proposed method after six months. The paper concludes in Section VIII with a discussion of results and suggestions for future work.

II. Related Work

There has been a growing research interest in device management strategies due to the increased urgency for maintaining a secure and functioning network. OS classification is a major facet of device management and as such has received considerable attention in the past few years [1].

The most widely used OS classification technique is packet inspection. Packet inspection relies on the idiosyncrasies in the packet creation process of different OSs. For example, the time-to-live (TTL) field of a packet is often used to identify Windows OSs (TTL=128) from both Apple and Android OSs (TTL=64) [7]. Packet inspection can also extract more comprehensive signatures from the payload of a packet to increase the fidelity of the classification that can be achieved. For example, the HTML User-Agent field can often be used to identify the exact version of a device's OS [2]. However, the widespread adoption of encryption standards has reduced the efficacy of techniques that rely on the inspection of payload contents.

There are several well-known OS classification tools that make use of packet inspection (e.g., Nmap [9], Xprobe2 [3], and p0f [16]). These techniques often show high classification performance as well as the ability to distinguish between distinct versions of the same OS. Nmap and Xprobe2 however both require the target device to be connected to the network during analysis. Therefore, these tools can not be used to perform an investigation into devices that are no longer present on the network. p0f can be used to perform a retrospective analysis of a device's OS given sufficient packet data; however, the storage and processing of packet data can lead to security and privacy concerns if it is insufficiently sanitised.

A device's OS can also be predicted through observing how its network traffic changes over time. For example, the frequency in which a device sends consecutive packets within a network communication can be indicative of its underlying OS [10], [13]. The challenge of such approaches is that there can be significant variation in a device's network characteristics due to the relative activity on the device. The optimal results are often observed when the device is under a heavy network load. Additionally, these techniques are difficult to scale for large networks given the added processing time required to construct their feature set.

The aim of this paper is to produce an OS classifier that uses only the bare minimum of information that would be available to a network administrator. Through restricting our analysis to just the IP addresses that a device affiliates with, we create an encryption invariant classification that is widely applicable to any TCP/IP network. In our previous work [12], we have shown the feasibility of affiliation graphs for OS classification. In this paper we provide an extension of this work through the utilisation of machine learning (ML) to identify relevant communities of interest for OS classification. Additionally, we further our research by: 1) examining the required monitoring length of a host device before an accurate prediction of its OS can be made; and, 2) investigating the longitudinal aspects of our proposed method over the course of six months.

III. Affiliation Graphs for OS Classification

The axiom of affiliation graphs is that there exists a reason for which actors are affiliated with certain communities [14]. Once this reason is known, it can be used to infer information about the actors affiliated with a particular community without any *a priori* knowledge of the actors themselves. Affiliation graphs have shown widespread use in sociological surveys for investigating the behavioural characteristics that motivate an individual to affiliate with a certain set of communities. The insight drawn in this paper is that the characteristics of a device can likewise be identified through the communities it affiliates with over the Internet.

Devices are now reliant on online infrastructure to manage, notify, and update the majority of the software they run. The software on a device can therefore often be reliably identified through monitoring the device's communication to its server-side dependencies. The benefit of using an affiliation graph to investigate these communications is the ease in which it can be constructed. To create an affiliation graph from network data, only the source and destination IP addresses are required from each network flow[1]. This results in a technique that is quick to populate and requires little storage. Additionally, this technique is also widely deployable as most network administrators have access to such flow records.

An example of an affiliation graph populated from 26 hours of the network traffic from eight devices is shown in Figure 2. It can be seen that the majority of communities that a device affiliates with over the span of 26 hours are owned by the manufacturers of their respective OSs.

To represent an affiliation graph to the ML classifier, a one-hot encoding of the communities that each device affiliated with was used.

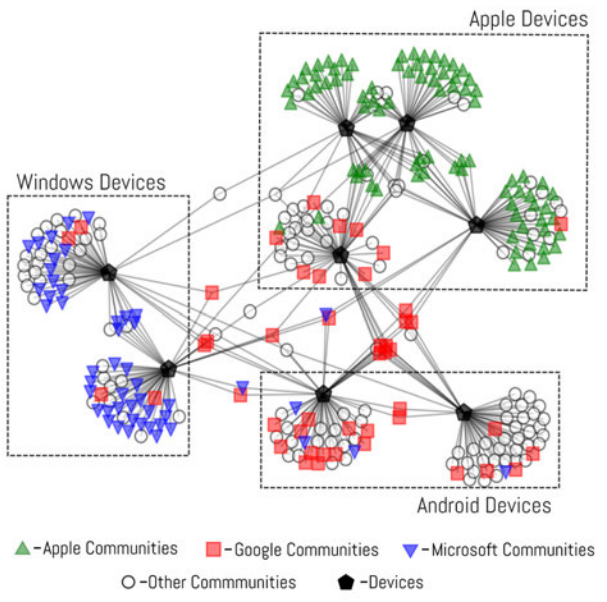


Fig. 2. An affiliation graph populated over 26 hours of network traffic for eight devices (Four Apple OS, two Windows OS, and two Android OS). Only the top 50 community affiliations for each device were depicted to increase readability.

[1]A network flow (or flow) is defined in this work as the bi-directional exchange of information between two devices over the Internet.

A. *Data Set – University Campus Network*

A large and realistic data set is required to adequately analyse the behavioural characteristics of different OSs. To provide this data set, network captures were taken on one of the University of Adelaide's wireless networks. This network was created to allow both staff and students to connect their wireless devices into the university's enterprise network. It therefore provides our analysis with insight into the realistic behaviour of current OSs.

The wireless network under analysis is a /23 subnet which allows for up to 510 distinct IP address to be allocated at any one time. An IP address however can be re-allocated to multiple distinct devices within the capture period. To account for this re-allocation of IP addresses, the address resolution protocol (ARP) log was used to identify when a device was allocated to a particular IP address.

The ground truth for the OSs on the network was acquired through dynamic host configuration protocol (DHCP) fingerprinting. DHCP fingerprinting allows for the identification of a host's OS through a signature-based analysis of the parameters it requests during its DHCP handshake. Devices with OSs that could not be adequately identified through DHCP fingerprinting were labelled as 'Unknown' and excluded from this analysis.

Three 26-hour captures were taken on the University's wireless network. These captures were taken three months apart on the 1st of March (t_0), August (t_1), and November (t_2) of 2019. The three month gap between captures was scheduled to examine the classification performance of our methodology over time. All subsequent analysis will be conducted on the t_0 capture except for Section VII (*Longitudinal Analysis*) which will be conducted on all three captures.

The distribution of OSs within the wireless network is shown in Table I. OSs were also grouped into an OS Family category. OS families are groups of OSs which share similar foundations either through being forked from the same code base or through a shared manufacturer. OSs within an OS

TABLE I
The distribution of operating systems on the university's wireless network. Network captures were taken on the 1st of May (t_0), August (t_1), and November (t_2) of 2019.

OS Family	OS	t_0	t_1	t_2
Apple	iOS	240	217	234
	OS X	91	77	84
Windows	Windows 10	26	26	21
	Windows 8/8.1	2	2	2
	Windows Vista/7	17	15	14
Android	Samsung (One UI)	37	26	31
	Huawei (EMUI)	19	15	10
	Oppo (ColorOS)	10	10	8
	Google (Stock)	1	2	2
	Xiaomi (MIUI)	3	2	2
	Other Android	51	50	50
Other	Linux (Ubuntu)	2	2	0
	Printer	1	3	2
	Unknown	181	151	172
Total		681	598	632

Family tend to share similar functionality and vulnerabilities. Therefore, classifying a host device's OS Family is still beneficial when the exact OS of the device cannot be determined.

In order to adequately train a ML classifier there must be enough samples to train from. Therefore, Windows 8/8.1, Google (Stock), and Xiaomi (MIUI) were removed from the analysis of OS classification as these classes only had three or fewer samples each. These classes however were still included when classifying the overall OS Family to provide a more diverse range of samples for the ML algorithm to learn from. Additionally, devices with 100 flows or fewer were excluded from this analysis as they were not connected to the network for long enough to exhibit characteristic behaviour.

IV. Communities for OS classification

The community set of the t_0 capture contained a total of $19,222$ distinct communities. However, not all of these communities will be relevant for OS classification. Therefore, a feature selection process must first be used to remove the redundant communities. Implementing a feature selection process before training an OS classifier has been shown to improve the overall performance that can be achieved [1]. Additionally, removing redundant communities can help direct further research into which communities reveal the most insight into OS classification.

Recursive feature elimination (RFE) [8] was used to perform feature selection on the set of communities within the t_0 capture. RFE iteratively trains a ML classifier with fewer features after each iteration. The features with the least importance to the overall classification performance will be selected for removal.

Ten trials of RFE were performed on the t_0 capture to identify the most significant communities for OS classification. For each trial, 80% of the capture was randomly selected for training an RF classifier using RFE. After training the RF classifier, the Macro and Micro F1-Scores [15] were evaluated on the remaining 20% of the capture to provide an overview of the classification performance at each iteration. Figure 3 provides a summary of the results from the ten trials of RFE. The classification performance was highest when the top 200 communities were used for classification. Fewer communities resulted in a classifier that is unable to properly distinguish between different OSs; whereas, additional communities increased the difficultly for the classifier to find appropriate models of best fit.

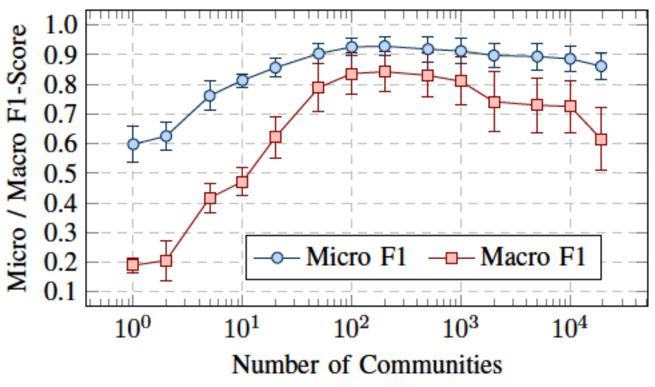


Fig. 3. Classification performance versus the number of communities used for OS classification. Recursive feature elimination (RFE) was used to reduce the number of communities in each iteration of training. Ten trials of RFE were trained on 80% of the t_0 capture. The mean and standard deviation of the Micro and Macro F1-Scores of the 20% validation set are shown for each iteration.

From the list of the top 200 communities for OS classification, it was found that 145 (72.5%) were communities owned by either Apple, Microsoft, or Google. This result was not surprising given these are the respective manufacturers of the three families of OSs under analysis. The communities within this set provide the services required to manage and update their respective OS. For example, a Microsoft community (52.229.171.202) provided Windows updates to its affiliated devices during the t_0 capture. A further analysis into the information that can be resolved from the community set can be found in our previous work [12].

The top 200 communities identified in this section were used as the feature set for all subsequent analysis.

V. OS Classification

The aim of OS classification is to acquire as much information about the OS running on a device as possible. Given this aim, OS classification has been divided into two categories:

1) **OS Family classification** aims to identify the family of OSs for which the target OS belongs (i.e., Apple, Windows, or Android). A device's OS Family often dictates the managerial and security practises which should be enforced for that device. OS Family classification is therefore beneficial for obtaining a general overall of the OS composition on a network.
2) **OS classification** aims to identify the distinct OS running on a device. OS classification allows for a greater level of insight into the specific OS on the device. OS classification is therefore often used when a more targeted analysis of a set of devices is required.

To examine the classification performance of our methodology, ten trials of an RF classifier were conducted on the t_0 capture. For each trial, an 80% training set and 20% validation

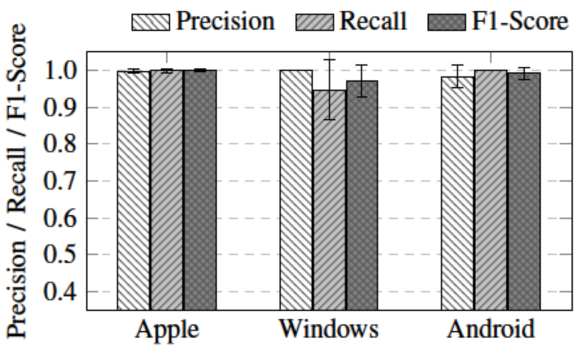


Fig. 4. Performance of OS Family classification. Ten trials of training an RF classifier were performed. In each trial 80% of the t_0 capture was randomly selected for training. The results show the mean and standard deviation of the precision, recall, and F1-Score of the classes within the 20% validation set.

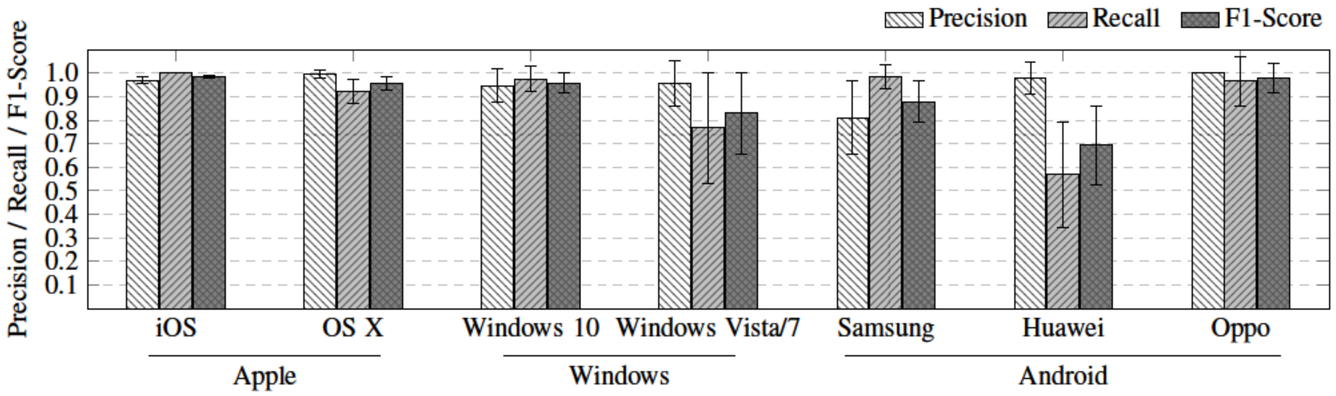


Fig. 5. Performance of OS classification. Ten trials of training an RF classifier were performed. In each trial 80% of the t_0 capture was randomly selected for training. The results show the mean and standard deviation of the precision, recall, and F1-Score of the classes within the 20% validation set.

set were randomly selected. The result of this training process for both OS Family and OS classification are given in Figures 4 and 5, respectively.

The overall average accuracy for both OS Family and OS classification were 99.3% and 94.6%, respectively. These results highlight that an accurate prediction of a device's OS can be made using only an easily obtainable feature set. This methodology is therefore well suited for large networks given the availability of the feature set used and the limited storage space required. However, the underlying assumption is that the host device is not actively obscuring its network behaviour. For example, a device could use a virtual private network (VPN) service to obscure the communities it affiliates with. This obfuscation would reduce the effectiveness of our methodology for OS classification. However, the resultant reduction in a device's community set could be used to detect which devices are using VPN services.

VI. Time Interval

The rise in popularity of portable devices (e.g., laptops, mobile phones, and tablets) has decreased the length of time for which a device is typically connected to a given network. It is therefore prudent to investigate how long a device is required to be monitored on a network for a reliable classification to be made.

The results presented in Sections IV and V were conducted on the entire t_0 capture. Therefore, up to 26 hours of network activity for each device could be utilised to perform OS classification. However, not all devices were active on the network for the entire capture period. Within the t_0 capture, the average time interval a device was active on the network was 8.67 ± 6.80 hours. The large variation in this time interval indicates that the classification made in the previous sections are not heavily dependent on the length of capture. However, these results do not give a good indication of the classification performance for devices which are only present on the network for a short period of time (e.g., less than an hour).

The network activity of the devices in the t_0 capture was iteratively divided into smaller time intervals to further examine the effect that the length of capture has on classification performance. For each time interval, ten trials were conducted using 80% of the t_0 capture to train an RF classifier and 20% to validate its performance. The result of these trials can be seen in Figure 6.

Figure 6 depicts a relatively stable classification performance for a time interval of one hour and above. 490 (98.4%) of the devices within the t_0 capture exceeded this time interval threshold. Therefore, this result supports the notion that the classifications made in Sections IV and V were not heavily dependent on the length of capture. However, a substantial decrease in classification performance was exhibited for time intervals of less than an hour. Therefore, we posit that on average it takes approximately one hour to construct a characteristic profile of a device's OS through only monitoring the IP addresses it communicates with over the Internet.

VII. Longitudinal Analysis

It is expected that the characteristic behaviour of an OS will change over time. For example, an OS may receive an update modifying the online services (and hence community IP addresses) that it communicates with. Conversely, an online service may modify the IP addresses it uses to communicate with its client-side devices. In both cases, the classification performance of this methodology will reduce as more modifications are made to an OS's online ecosystem. A longitudinal

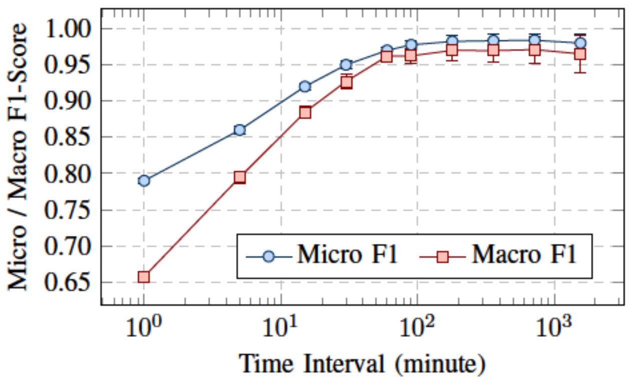


Fig. 6. Classification performance versus the time interval used for analysis. Ten RF classifiers were trained for each time interval. For each classifier, 80% of the t_0 capture was used for training with 20% reserved for validation. The mean and standard deviation of the Micro and Macro F1-Scores of the 20% validation set are shown for each time interval.

analysis was therefore conducted to examine the length in which the classification performance of this methodology is expected to remain significant.

Two additional captures of the same university network were taken three (t_1) and six (t_2) months after the t_0 capture. These captures were taken to provide an insight into the changes of an OS's online ecosystem over the course of six months. An RF classifier was trained on 80% of the t_0 capture and tested on 20% of the t_0 capture and 100% of the t_1 and t_2 captures. Table II summarises the OS Family and OS classification performance on the test set of all three captures.

Table II indicates a negative correlation between the classification performance and the time period between the training and testing captures. This result supports the expectation that an OS's online ecosystem will change over time causing a degradation in this methodology's classification performance. However, there was a less than 10% decrease in the overall classification performance (Micro F1-Score) for both OS Family and OS classification after six months. Therefore, the degradation in classification performance may still be tolerable for acquiring an initial overview of the OS composition on a network.

There was a large variation between the classification performance of individual classes over the six month period. Both Apple and Android OSs showed a relatively small decrease in their individual F1-Scores (3% and 7%, respectively) when evaluated on the t_2 capture. However, a 29% decrease in the F1-Score of Windows OSs was seen for the same period. There are multiple factors which may have affected the classification performance of this methodology on Windows OSs. For example, multiple updates to the Windows OS were released during the six month period between the t_0 and t_2 captures. It is therefore recommended that this methodology be retrained every few months or after a significant update to an OS is released.

VIII. Conclusion and Future Work

In this paper a minimalist approach to OS classification was introduced. This minimalist approach illustrated that an accurate OS classifier can be trained using only the IP addresses that a device communicates with via the Internet. In particular, we show that only a set of 200 community IP addresses were required to classify distinct OSs. Furthermore, we identified that on average it takes approximately one hour to construct a characteristic profile of a device's OS through only monitoring the IP addresses it communicates with over the Internet. Lastly, we found that the classification performance of this methodology will deteriorate at different rates for each OS. This deterioration can be mitigated by retraining the classifier every few months or after a significant update to an OS is released.

In future work we aim to provide a broader analysis on utilising affiliation graphs for device characterisation. In particular, we aim to utilise unsupervised machine learning to identify clusters of devices on a network which exhibit similar behavioural characteristics.

TABLE II
Longitudinal analysis of OS Family and OS classification performance.

Performance Metric (F1-Score)	t_0 01/05/2019	t_1 01/08/2019	t_2 01/11/2019
OS Family Classification			
Micro	0.98	0.94	0.92
Macro	0.97	0.93	0.84
OS Classification			
Micro	0.90	0.83	0.81
Macro	0.78	0.64	0.61

References

[1] Ahmet Aksoy, Louis Sushil, and Mehmet Haid Gunes. Operating system fingerprinting via automated network traffic analysis. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2502–2509, 2017.

[2] Blake Anderson and David McGrew. OS fingerprinting: New techniques and a study of information gain and obfuscation. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2017.

[3] Ofir Arkin, Fyodor Yarochkin, and Meder Kydyraliev. The present and future of Xprobe2: The next generation of active operating system fingerprinting. *SYS-Security Group*, 2003.

[4] Deepali Arora, Kin Fun Li, and Alex Loffler. Big data analytics for classification of network enabled devices. In *Advanced Information Networking and Applications Workshops (WAINA)*, pages 708–713, 2016.

[5] Stephen Borgatti and Daniel Halgin. Analyzing affiliation networks. *The Sage Handbook of Social Network Analysis*, 2011.

[6] Ronald L Breiger. The duality of persons and groups. *Social Forces*, 53(2):181–190, 1974.

[7] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. OS fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 173–180. ACM, 2014.

[8] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[9] Gordon Fyodor Lyon. *Nmap Network Scanning: The official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[10] Nikunj Malik, Jayanarayan Chandramouli, Prahlad Suresh, Kevin Fairbanks, Lanier Watkins, and William H. Robinson. Using network traffic to verify mobile device forensic artifacts. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 114–119, 2017.

[11] Jeremy Merrill. Liberal, moderate or conservative? see how Facebook labels you, 2016. [Online]. Available: https://www.nytimes.com/2016/08/24/us/politics/facebook-ads-politics.html. [Accessed: September 2019].

[12] Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim. Characterising network-connected devices using affiliation graphs. *2020 IEEE/IFIP Network Operations and Management Symposium*, 2020.

[13] Nicholas Ruffing, Ye Zhu, Rudy Libertini, Yong Guan, and Riccardo Bettati. Smartphone reconnaissance: Operating system identification. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1086–1091, 2016.

[14] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *IEEE 12th International Conference on Data Mining*, pages 1170–1175. IEEE, 2012.

[15] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[16] Michal Zalewski. p0f v3, 2014. [Online]. Available: https://lcamtuf.coredump.cx/p0f3/. [Accessed: March 2020].

Chapter 5

Graph-Based Machine Learning for Network Reconnaissance

This chapter provides two novel graph-based machine learning (ML) techniques for conducting passive network reconnaissance. Graph-based ML was investigated to address the following three key limitations of applying conventional ML techniques for the analysis of the bipartite point-of-analysis (PoA):

1. **Static Analysis:** Conventional ML must be retrained in the event that a new community (i.e., Internet service) is detected. This limitation poses a severe restriction for the analysis of a TCP/IP network as Internet services are in constant flux. Furthermore, supervised ML (such as proposed in Chapter 4, Paper 4.4) may be difficult to retrain in dynamic network environments due to an absence of labelled data.

2. **Feature Space:** The number of features used by a conventional ML technique is dependent on the number of observed communities[1]. The number of communities used within a large TCP/IP network would result in a feature set that would be easily overfit. For example, over 500 thousand unique communities were observed from a 26-hour observation of a university's enterprise network [127].

[1]Feature selection (such as conducted in Chapter 4, Paper 4.4) could be utilised to reduce the number of features; however, all communities must be present for the initial feature selection process.

3. **Community Similarity:** Each community is assumed to be an independent feature. This limitation is sub-optimal as Internet services are often distributed in which distinct communities facilitate the same Internet service. In contrast, graph-based ML quantifies community similarity through the analysis of the structural similarity of a community's neighbourhood set.

The three listed limitations would restrict the deployment of the bipartite PoA for the analysis of large-scale TCP/IP networks. In this chapter, we design novel graph-based ML techniques to address these limitations.

Two papers are presented within this chapter:

Paper 5.1 - "*Detecting Botnet Victims Through Graph-Based Machine Learning*" provides an analysis of graph embedding techniques for the intrusion detection objective. This paper provides an extension to GraphSAGE [87] to detect botnet infrastructure within a TCP/IP network. The axiom of this approach is that the connection profile of a device can be used to identify the command and control (C2) botnet infrastructure.

Paper 5.2 - "*Enhancing Situational Awareness in Encrypted Networks Using Graph-Based Machine Learning*" devises a novel graph embedding technique—bipartite graph embeddings (BGE)—designed to provide a comprehensive framework for conducting passive network reconnaissance through the bipartite PoA. This technique is extensively evaluated on application areas relevant for the network reconnaissance domain (i.e., service prediction, device characterisation, and network visualisation).

The key contribution of this chapter is as follows:

1. (**Paper 5.1**) We provide an extension to the well-known GraphSAGE technique [87] to analyse a bipartite graph. We designated this extension as BiSAGE and evaluate its use within the intrusion detection objective. In particular, we show that BiSAGE improves the ability to detect botnet infrastructure in comparison to its predecessor. Furthermore, we show that BiSAGE can accurately identify botnet infrastructure

without requiring any labelled samples of botnet activity. This technique could therefore be utilised to detect novel variants of botnet behaviour and support long-term deployment (CN4).

2. (**Paper 5.2**) We design bipartite graph embeddings (BGE). BGE is the first graph embedding technique that enables the real-time analysis of a large enterprise network. We show that BGE remains effective under partial network observation and efficiently scales for the analysis of networks containing hundreds of thousands of devices. We provide BGE as packaged tool[2] that can be used to generate insight into bipartite graphs used in other application domains (e.g., recommendation and citation networks).

3. (**Paper 5.2**) We validate that BGE satisfies all four criteria for widespread deployment within realistic network conditions (CN1-4). In particular, BGE is an unsupervised ML technique that can be easily updated in response to changes in the network environment and enforces an open-world assumption (CN4). Furthermore, we show that the embeddings produced by BGE can be reused to satisfy distinct network reconnaissance objectives (e.g., device and application characterisation). BGE thus provides an objective independent methodology for analysing a TCP/IP network through the bipartite PoA.

4. (**Paper 5.2**) We provide a novel edge sampling strategy—transient edge sampling (TES)—to enable BGE for the analysis of large-scale and highly-dynamic graphs. We show that TES provides an order-of-magnitude reduction in the training time of BGE when compared to current sampling strategies (i.e., edge sampling and random walks). We validate that the reduction in training time does not reduce the fidelity of the resultant graph embeddings.

[2]https://github.com/MillarK-UofA/bipartite_graph_embeddings

Statement of Authorship

Title of Paper	Detecting Botnet Victims Through Graph-Based Machine Learning
Publication Status	[✓ Published] ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	K. Millar, L. Simpson, A. Cheng, H. G. Chew, and C.-C. Lim, "Detecting Botnet Victims Through Graph-Based Machine Learning," in International Conference on Machine Learning and Cybernetics (ICMLC), 2021: IEEE, 2021, pp. 1-6.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	60%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Lachlan Simpson		
Contribution to the Paper	Assisted in design, analysis, and interpretation of data. Assisted in drafting the manuscript.		
Signature		Date	15/09/22

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22.

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	5/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Detecting Botnet Victims Through Graph-Based Machine Learning

Kyle Millar[1], Lachlan Simpson[2], Adriel Cheng[1,3], Hong Gunn Chew[1], Cheng-Chew Lim[1]

[1]School of Electrical and Electronic Engineering, The University of Adelaide, Australia
[2]School of Mathematical Sciences, The University of Adelaide, Australia
[3]Cyber and Electronic Warfare Division, Defence Science & Technology Group, Australia
E-MAIL: {kyle.millar, lachlan.simpson, adriel.cheng, honggunn.chew, cheng.lim}@adelaide.edu.au[1,2]
E-MAIL: adriel.cheng@dst.defence.gov.au[3]

Abstract:

Botnets are one of the most devastating cybersecurity threats to modern organisations. A botnet is a distributed network of compromised devices that is leveraged to perform various malicious operations over the internet. The recent proliferation of unabated botnet activity has necessitated the investigation of novel botnet detection strategies. In this paper, we introduce BiSAGE; a graph-based machine learning technique capable of detecting the compromised hosts (*bot victims*) operating on a network. The advantage of our approach is that a bot victim can be detected not only through its actions but also through the actions of the devices it communicates with; an intrinsic characteristic of botnet activity.

We provide an empirical evaluation of BiSAGE on CSE-CIC-IDS2018; a comprehensive dataset for evaluating intrusion detection systems. We show that BiSAGE is able to accurately identify bot victims without requiring any labelled samples of botnet activity.

Keywords:

Cybersecurity, graph-based machine learning, botnet detection.

1. Introduction

A botnet is a distributed network of compromised devices that are controlled by a malicious actor. Botnets leverage their network of compromised devices to perform various malicious operations over the internet. Such operations could include devastating attacks such as a distributed denial of service (DDOS), phishing campaigns, as well as exfiltrating personal information from the compromised devices [1].

The proliferation and severity of botnet activity has become one of the most serious cybersecurity threats faced by modern organisations [15, 2]. A major factor in the recent spread of botnet activity has been the introduction of internet-of-things (IoT) devices due to their rapid adoption and known security vulnerabilities [9]. Furthermore, the rising price of cryptocurrency has further incentivised the spread of botnets for their use as commodity malware and distributed cryptominers [12]. It is evident that there will be a continued rise in botnet activity for the foreseeable future. It is therefore necessary to investigate novel detection strategies to mitigate the damage caused by botnet activity.

Figure 1 depicts a typical botnet infrastructure. The controller of the botnet (referred to as the *bot master*) creates a set of command and control (C2) servers. The C2 servers relay communications between the bot master and the compromised hosts on the botnet (referred to as the *bot victims*).

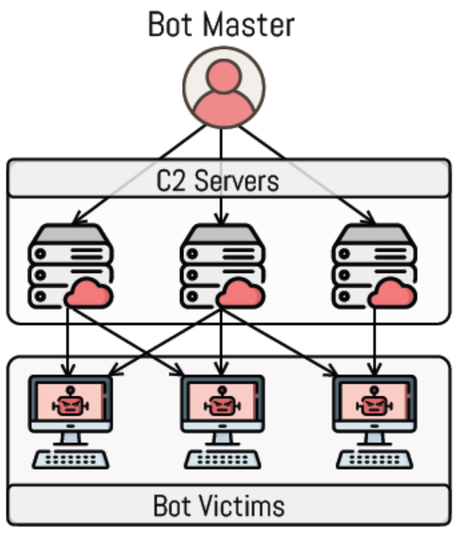


FIGURE 1. A typical botnet infrastructure. The command and control (C2) servers relay commands from the bot master to the bot victim.

A host becomes a bot victim through various means. Typically a bot victim is first compromised through either opening a malicious email attachment or visiting a compromised website [2]. Once compromised, a bot victim is then instructed by the bot master to perform various malicious operations.

Current botnet detection solutions attempt to detect a bot victim based on its own behaviour. Malicious behaviour exhibited by a bot victim however can become obfuscated by benign user activity on the host. To address this limitation, we pose the use of graph-based machine learning to detect a bot victim through not only its behaviour but the behaviour of the devices it communicates with. Our approach exploits the inherent property that bot victims must communicate within the bot infrastructure to receive commands or exfiltrate data.

The contribution of this work is two-fold:

1. We introduce BiSAGE; a modification of the GraphSAGE algorithm [5] that has been specifically tailored for the detection of bot victims. We show that BiSAGE provides a significant improvement over GraphSAGE for the task of bot victim detection.

2. We evaluate the use of BiSAGE on CSE-CIC-IDS2018 [11]; a publicly available dataset for evaluating intrusion detection systems. We show that BiSAGE is able to accurately identify bot victims without requiring any labelled samples of botnet activity. This methodology can therefore be utilised to detect new variants of botnet behaviour.

The remainder of this paper is structured as follows: A brief survey of related work in botnet detection is given in Section 2. Section 3 introduces BiSAGE; a technique for detecting botnet activity through graph-based machine learning. Section 4 outlines the dataset used for our analysis. Section 5 provides an empirical analysis of the efficacy of the proposed technique. The paper concludes in Section 6 with a discussion of results and suggestions for future work.

2. Related Work

Recent advancements in botnet detection have largely stemmed from the use of machine learning (ML) to classify a botnet's communication channels [13]. This approach is divided into two main categories - flow-based detection and host-based detection.

Flow-based detection techniques [14, 16, 8, 4, 6] attempt to classify whether individual streams of network traffic are associated with botnet activity. Such techniques typically rely on the uniformity of botnet communications.

Bot masters must maintain a connection to their bot victims to issue new commands and exfiltrate data. This behaviour typically generates communication patterns that exhibit a higher degree of uniformity than is characteristic of benign network traffic.

Flow-based detection characterises the uniformity within individual flows of traffic. This basis of detection therefore does not examine the inter-flow dynamics indicative of botnet behaviour [7]. To address this limitation, recent studies have investigated the detection of botnet communications at the host level.

Host-based detection techniques [7] classify whether a host is associated with a botnet based on its overall communications. Such techniques are able to detect longitudinal behaviour of a botnet that may not be identifiable through a flow-based analysis alone. Host-based detection techniques however are singularly faceted in that they do not consider the interplay between devices on a network.

The distinctive characteristic of a botnet is that the bot victims are not isolated; they are part of larger network of C2 servers and additional bot victims. A technique that classifies each host individually is therefore missing a crucial element that comprises a botnet. To address this limitation, we pose BiSAGE; a graph-based machine learning technique that is able to detect a bot victim through not only its actions but the actions of the devices it communicates with.

3. BiSAGE

The BiSAGE technique is divided into two main components. Firstly, the flow features for each host under analysis are aggregated over the capture period (Section 3.1). Secondly, graph-based machine learning is used to aggregate a host's flow features with the flow features of the devices it communicates with (Section 3.2).

3.1 Flow Features

The efficacy of a botnet detection system is reliant on the features under analysis [1]. Previous investigations have extensively enumerated the features most indicative of flow-based botnet behaviour. A summary of 10 flow-based features commonly used for botnet detection is provided in Table 1.

To perform a host-based analysis, the flow features provided in Table 1 must be aggregated for each host. Aggregating the flow statistics for a given host however is non-trivial. The diversity of behaviour exhibited by each host must be accounted for in order to observe meaningful statistics. In particular, bimodal distributions were found when aggregating the flow statistics described in Table 1. Such distributions therefore reduce the information gained through common aggregation functions such as mean and standard deviation.

Frequency distributions were utilised to account for the diversity in host behaviour when aggregating flow-based statistics. A frequency distribution measures the frequency of occurrence for values within set intervals. The aggregation metric for each feature was set as the proportion of a host's network traffic that was represented by the two most frequent intervals. This aggregation metric therefore provides a measure of the uniformity of host's network traffic in the presence of known bimodal distributions. Furthermore, the proposed aggregation function does not depend on the absolute values for each feature. This property is particularly beneficial as methods that rely on the absolute value of a feature are unlikely to generalise to new botnet variants.

The difficulty in detecting a bot victim through its feature vector alone is that the malicious behaviour can become obfuscated by benign user activity on the host. To address this limitation, we make use of the inherent botnet infrastructure. Bot victims must communicate with their C2 servers to receive commands and exfiltrate data. Therefore, the servers a host communicates with provides an additional layer of information when detecting a bot victim. Furthermore, additional bot victims are likely to also contact the same C2 servers. Therefore, the hosts which share servers in common with the bot victim can also be beneficial for botnet detection.

Methods to aggregate the feature vector of the hosts on an enterprise network and the servers that they communicate with must therefore be investigated.

TABLE 1. A summary of 10 features commonly used in flow-based botnet detection. Several studies were noted to combine the 'Forward' and 'Backward' features into a single metric; in this study, we have refrained from this aggregation to mitigate the loss of information.

Features	Reference	Description
Forward_Pkt_Len	[14, 16, 8, 4]	Average length of the packets sent to the destination IP.
Backward_Pkt_Len	[14, 16, 8, 4]	Average length of the packets sent to the source IP.
Forward_Pkt_Rate	[14, 16, 8, 4]	Average packets per second sent to the destination IP.
Backward_Pkt_Rate	[14, 16, 8, 4]	Average packets per second sent to the source IP.
Forward_Pkts	[14, 16, 8, 6]	Total number of packets sent to the destination IP.
Backward_Pkts	[14, 16, 8, 6]	Total number of packets sent to the source IP.
Forward_Bytes	[14, 6]	Total bytes sent to the destination IP.
Backward_Bytes	[14, 6]	Total bytes sent to the source IP.
Forward_IAT	[16]	Average inter-arrival time between packets sent to the Destination IP.
Backward_IAT	[16]	Average inter-arrival time between packets sent to the Source IP.

3.2 Neighbourhood Aggregation

A bipartite graph was used to represent the communications between the hosts on the network (*actors*) and the servers (*communities*) in which they communicate with. An example of the bipartite graph representation is illustrated in Figure 2 (a). A bipartite graph was used as it allows for an intrinsic representation of the services used by the hosts on an enterprise network [10].

The neighbourhood of a vertex, $\Gamma(v)$, is the set of all vertices which share an edge with vertex, v. The neighbourhood of a vertex therefore allows for easy resolution of the servers affiliated with a host, and the hosts affiliated with a server. As discussed in Section 3.1, aggregating the feature vectors in the neighbourhood of a host and its affiliated servers is likely to provide additional insight when detecting a bot victim.

Graph convolutions neural networks (GCNs), such as GraphSAGE [5], have been designed to aggregate feature vectors from the neighbourhood of a vertex. GCNs are however typically applied to homogeneous graphs. While a homogeneous GCN could be applied to evaluate a bipartite graph, it would not benefit from the unique properties that define bipartite graphs.

To perform neighbourhood aggregation in our analysis, we pose BiSAGE; a modification of the GraphSAGE algorithm for the evaluation of bipartite graphs. In particular, BiSAGE exploits the unique property of a bipartite graph that edges exists only between vertices of the opposing set. This property allows for two improvements of the GraphSAGE algorithm to be made:

1. The creation of a specialised random walk for traversing a bipartite graph (Algorithm 1). The bipartite random walk is initialised from an actor vertex and transverses its affiliated communities. This random walk has been specifically designed to aggregate information into the set of actor vertices for the purpose of bot victim detection.

2. The enforcement of distinct aggregation layers between the actor$\rightarrow$community and community$\rightarrow$actor relationships. This improvement allows for better defined aggregations functions to be learned during training. The improvement made to GraphSAGE has been shown in Algorithm 2. In particular, line 3 was introduced to alternate between the actor and community vertices within a bipartite graph.

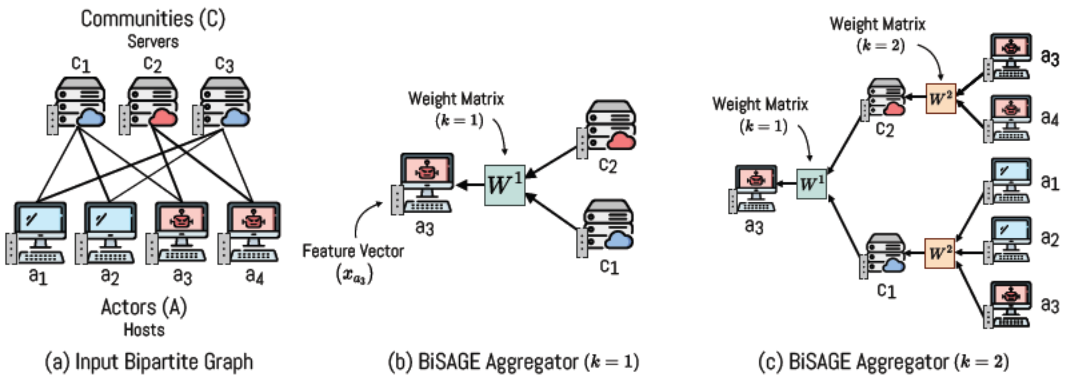


FIGURE 2. The feature aggregation methodology used within BiSAGE. Benign and botnet hosts has been highlighted in blue and red, respectively. (a) the bipartite representation of the communication between hosts on an enterprise network (*actors*) and the servers in which they communicate (*communities*). (b) BiSAGE aggregation at depth ($k = 1$). (c) BiSAGE aggregation at depth ($k = 2$).

The improvements made to GraphSAGE allow for a better feature aggregation for the detection of bot victims on an enterprise network. This feature aggregation is evaluated in Section 5 at two aggregation depths (k). The first aggregation depth ($k = 1$) detects a bot victim through its feature vectors and the feature vectors of the servers it communicates with (Figure 2 (b)). The second aggregation depth ($k = 2$) detects a bot victim through aggregating the feature vectors of the devices which share severs in common with a bot victim (Figure 2 (c)).

The output of BiSAGE is a set of embeddings for both the actor and community vertices within the bipartite graph. The embedding for each vertex represents the feature vector of its associated host aggregated with the feature vectors of the hosts it communicates with. The embeddings generated by BiSAGE can then be evaluated using traditional machine learning to detect the bot victims operating on an enterprise network.

Algorithm 1: Bipartite Random Walk

```
1  Set walks per node ($r$) and length of walk ($l$)
2  Initialise empty array, $walks$
3  for vertex $a$ in $A$ do
4      for $iter_{walk} \leftarrow 1$ to $r$ do
5          Initialise $walk$ to $[a]$
6          Select community vertex, $c \in \Gamma(a)$
7          Append $c$ to $walk$
8          for $iter_{step} \leftarrow 1$ to $l$ do
9              Select an actor vertex, $a' \in \Gamma(walk[-1])$
10             Select a community vertex, $c' \in \Gamma(a')$
11             Append $c'$ to $walk$
12     Append $walk$ to $walks$
13 return $walks$
```

where the last vertex in the walk is designated as $walk[-1]$. Furthermore, the vertices are selected from a given neighbourhood based on the sampling criteria defined in the Node2Vec algorithm [3].

4. Dataset

BiSAGE was evaluated on CSE-CIC-IDS2018 [11]; a comprehensive dataset for evaluating intrusion detection systems. The CSE-CIC-IDS2018 botnet dataset provides the network traffic of an enterprise network over a 24-hour period. Within this capture period, two distinct botnet variants were deployed, Zeus and Ares. Both botnets utilised the same C2 server to communicate with their bot victims. To investigate a more realistic botnet infrastructure, we further distributed the activity of the C2 server among five distinct community servers. This allowed for the analysis of a distributed botnet infrastructure.

In this paper, we restrict our analysis to five subnets within the CSE-CIC-IDS2018 enterprise network. This produces an analysis of 418 (98%) benign hosts and 10 (2%) bot victims. Furthermore, a total of 3,883 servers were contacted by the hosts on the enterprise network within the capture period; 5 (0.1%) of which were C2 servers.

Algorithm 2: BiSAGE Aggregation

```
1  $h_v^0 \leftarrow x_v, \forall v \in A \cup C$
2  for $k \leftarrow 1 \ldots K$ do
3      $N \leftarrow A$ if $k$ is odd; else, $N \leftarrow C$
4      for $n \in N$ do
5          $h_{\Gamma(n)}^k \leftarrow MEAN(\{h_u^{k-1}, \forall u \in \Gamma(n)\})$
6          $h_n^k \leftarrow \sigma(W^k \cdot CONCAT(h_n^{k-1}, h_{\Gamma(n)}^k))$
7      $h_n^k \leftarrow h_n^k / ||h_n^k||_2, \forall n \in N$
8  return $z_v \leftarrow h_v^K, \forall v \in A \cup C$
```

where A is the set of actor vertices; C is the set of community vertices; x_v is the feature vector of vertex, v; K is the aggregation depth; σ is a non-linear activation function; W^k is the weight matrix at aggregation depth k; and z_v is the final embedding generated for vertex, v.

TABLE 2. The Macro-F1 score obtained by a one-class SVM trained on the BiSAGE embeddings. The one-class SVM was trained only on the benign host samples. The percentage of benign host samples used for training is indicated in the columns of Table 2. For comparison, the Macro-F1 score obtained on the feature vectors of the hosts and the embeddings generated by Node2vec [3] and GraphSAGE [5] have also been provided.

Technique	Analysis	10%	20%	30%	40%	50%	60%	70%	80%	90%
Features	features	0.579	0.640	0.675	0.694	0.721	0.716	0.730	0.785	0.802
Node2Vec	structure	0.135	0.308	0.399	0.449	0.509	0.536	0.621	0.673	0.795
GraphSAGE ($k = 1$)	features + structure	0.518	0.550	0.570	0.625	0.633	0.638	0.674	0.689	0.694
BiSAGE ($k = 1$)	features + structure	0.606	0.662	0.733	**0.800**	0.809	0.853	**0.889**	**0.938**	**0.961**
GraphSAGE ($k = 2$)	features + structure	0.411	0.486	0.533	0.577	0.598	0.648	0.670	0.723	0.752
BiSAGE ($k = 2$)	features + structure	**0.708**	**0.725**	**0.780**	0.754	**0.852**	**0.872**	0.819	0.775	0.783

5. Results

One-class SVM was used to evaluate the embeddings generated by BiSAGE. One-class classification was selected as the performance metric as it allows for the detection of unknown malicious activity. This methodology can therefore be utilised to detect new variants of botnet behaviour.

Table 2 reports the classification performance of a one-class SVM model trained on the embeddings generated by BiSAGE. For comparison, a one-class SVM was also evaluated on the feature vectors of the hosts and the embeddings generated by Node2Vec [3] and GraphSAGE [5]. The Node2Vec algorithm was selected to investigate the information contained in the structure of the bipartite graph alone.

5.1 Features vs. Structure

The classification performance of the feature vectors of the hosts and the graph structure (Node2Vec) is shown in Table 2. It is seen that both evaluations are able to produce a reasonable boundary between the benign hosts and the bot victims. However, there is a significant improvement when using the hosts' feature vectors. This result is due to the specific curation of feature vectors for the purpose of botnet detection. The utilisation of the graph structure alone could still be beneficial when specific features of each host cannot be identified.

5.2 Feature Aggregation

GraphSAGE was evaluated to investigate whether the classification performance of the feature and structural analyses would be improved by their combined evaluation. The results of GraphSAGE in Table 2 show that the combined analysis performs worse than classifying the feature vectors of the actors themselves. This result shows that simply applying the GraphSAGE algorithm for the detection of bot victims is insufficient.

The application of GraphSAGE implicitly states that the graph under analysis is homogeneous. This assumption allows for relationships to exist between any two vertices within the graph. This assumption therefore does not delineate between the explicit relationships between the actor and community vertices, and the implicit relationships between the actors themselves. The extensions made within BiSAGE explicitly separate these relationships and thus can produce more meaningful embeddings on a bipartite graph.

5.3 GraphSAGE vs. BiSAGE

The comparison between the BiSAGE and GraphSAGE algorithms is shown Table 2. BiSAGE shows a significant improvement over GraphSAGE for both aggregation depths. This improvement is due to the tailoring of BiSAGE for the unique properties of a bipartite graph. Creating distinct aggregation layers for these relationships allows for a more efficient propagation of information between the actor and community sets. Furthermore, a ten times reduction in the training of GraphSAGE was achieved through the implementation of the bipartite random walk algorithm (Algorithm 1).

BiSAGE and GraphSAGE were both evaluated at an aggregation depth of $k = 1$ and $k = 2$. An overall improvement in classification performance was shown when using an aggregation depth of $k = 1$. This result is due to the specific application domain of bot victim classification. An aggregation depth of $k = 1$ attempts to classify a bot victim based on its feature vector and the feature vectors of the servers it affiliates with. As the C2 servers perform predominately malicious actions, their feature vectors would prove to be more informative than the feature vectors of the bot victims themselves.

6. Conclusion and Future Work

In this paper, a technique for detecting bot victims on an enterprise network was provided. The technique (BiSAGE) utilised graph-based machine learning to detect a bot victim through both its actions and the actions of the devices it communicates with. BiSAGE was evaluated on the CSE-CIC-IDS2018 dataset; a publicly available dataset for evaluating intrusion detection systems. BiSAGE was shown to be able to accurately identify bot victims without requiring any labelled samples of botnet activity. This methodology could therefore be utilised to detect new variants of botnet behaviour.

In future work, the application of graph-based machine learning will be investigated for use on a peer-to-peer (P2P) botnet infrastructure. This application would allow for more diverse botnet variants to be identified.

Acknowledgements

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

References

[1] ENISA threat landscape 2020 — botnet. [Online] Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-botnet/at_download/fullReport Accessed: 24 August 2021.

[2] Nathan Goodman. A survey of advances in botnet technologies. *arXiv preprint arXiv:1702.01132*, 2017.

[3] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[4] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *17th USENIX Security Symposium*, 2008.

[5] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[6] Rafał Kozik and Michał Choraś. Pattern extraction algorithm for netflow-based botnet activities detection. *Security and Communication Networks*, 2017.

[7] Rafał Kozik, Marek Pawlicki, and Michał Choraś. Cost-sensitive distributed machine learning for netflow-based botnet activity detection. *Security and Communication Networks*, 2018.

[8] Wen-Hwa. Liao and Chia-Ching Chang. Peer to peer botnet detection using data mining scheme. In *International Conference on Internet Technology and Applications*, pages 1–4, 2010.

[9] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized IoT devices using machine learning techniques. *arXiv preprint arXiv:1709.04647*, 2017.

[10] Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim. Operating system classification: A minimalist approach. In *International Conference on Machine Learning and Cybernetics*, pages 143–150, 2020.

[11] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *4th International Conference on Information Systems Security and Privacy*, volume 1, pages 108–116, 2018.

[12] Karl Sigler. Crypto-jacking: How cyber-criminals are exploiting the crypto-currency boom. *Computer Fraud & Security*, 2018(9):12–14, 2018.

[13] Simon Nam Thanh Vu, Mads Stege, Peter Issam El-Habr, Jesper Bang, and Nicola Dragoni. A survey on botnets: Incentives, evolution, detection and current trends. *Future Internet*, 13(8):198, 2021.

[14] Chun-Yu Wang, Chi-Lung Ou, Yu-En Zhang, Feng-Min Cho, Pin-Hao Chen, Jyh-Biau Chang, and Ce-Kuen Shieh. Botcluster: A session-based P2P botnet clustering system on NetFlow. *Computer Networks*, 145:175–189, 2018.

[15] Wei Wang, Yaoyao Shang, Yongzhong He, Yidong Li, and Jiqiang Liu. Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. *Information Sciences*, 511:284–296, 2020.

[16] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, 2013.

Statement of Authorship

Title of Paper	Enhancing Situational Awareness in Encrypted Networks Using Graph-Based Machine Learning
Publication Status	☐ Published ☐ Accepted for Publication [✓ Submitted for Publication] ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	submitted to IEEE Transactions on Network and Service Management.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Enhancing Situational Awareness in Encrypted Networks Using Graph-Based Machine Learning

Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim

***Abstract*—Situational awareness within an enterprise network is fundamental for network administrators and security analysts alike. The knowledge of what devices are on a network—and the purpose of their use—facilitates improved quality of service and allows for the identification of devices that may pose a security risk. The complexity of providing situational awareness within an enterprise network has been greatly exacerbated by the introduction of widespread encryption standards and bring-your-own (BYO) user devices. In this paper, we provide bipartite graph embeddings (BGE), a graph-based machine learning solution that enables situational awareness through analysing only the IP address field of network traffic. BGE therefore relies solely on easily extractable information, is invariant to encryption, and can be universally applied to any TCP/IP network configuration.**

We evaluate the applicability of current state-of-the-art graph embeddings techniques for the analysis of a university campus network. We identify that such techniques have severe limitations when applied for the analysis of large-scale and dynamic network environments. We provide BGE to address these limitations by exploiting the unique structural properties of a bipartite graph. We show that BGE can be directly applied for various device management tasks such as operating system classification and service prediction. Furthermore, we show that BGE remains effective under partial network observation and efficiently scales for the analysis of enterprise networks that consist of hundreds of thousands of devices.

***Index Terms*—Operational analytics, network situational awareness, knowledge representation, device management, service prediction, graph-based machine learning, neural networks.**

I. Introduction

NETWORK situational awareness is fundamental to combat the challenges of modern network security [1]. Situational awareness is the ability to identify, process, and comprehend the critical elements of a given environment [2]. Network situational awareness[1] provides the necessary insight to identify a network's vulnerabilities and facilitates faster response times in the event of a cyber-attack [3].

A central component of situational awareness is device management; the knowledge of what devices are operating on a network and the purpose of their use. Effective device management allows for the forecasting of future network utilisation and aids in identifying potential security vulnerabilities [4], [5].

Recent years have experienced a fundamental increase in the complexity of performing device management. This rising complexity has become unavoidable due to the sheer number and diversity of devices operating on modern networks [6], [7]. Additionally, device management has been further complicated by the introduction of internet-of-things (IoT) [8], bring-your-own device (BYOD) polices [9], and widespread encryption standards [10]–[12]. These complexities have introduced novel challenges in performing effective device management and thus have obfuscated the situational awareness that can be achieved within enterprise networks.

Current device management solutions typically rely on the use of fingerprint databases to identify the devices on a network (e.g., Nmap [13], DHCP fingerprinting [14], and p0f [15]). These methods have provided an intuitive and effective way of performing device management for many years. However, the efficacy of these approaches is declining due to three main factors:

1) **Volatility**: The rate in which new devices—and updates to their software—are released to market has drastically increased the management complexity of a fingerprint database [16].
2) **Encryption**: The widespread use of encryption has reduced the number of unique identifiers available for the creation of a device's fingerprint [10]–[12].
3) **Availability**: The availability of the identifiers used to create a device's fingerprint is dependent on the monitoring procedure of the network under analysis. Techniques that rely on a large number of unique identifiers to create a device's fingerprint are often too restrictive to deploy across diverse TCP/IP networks [17][2].

In this paper, we provide a technique to identify a device's characteristics through monitoring only the IP addresses it affiliates with over the Internet. This technique is therefore invariant to encryption and can be universally applied to any TCP/IP network configuration. Furthermore, we pose a graph-based machine learning approach to eliminate the reliance on maintaining a fingerprint database.

The proposed technique exploits the newfound reliance between a device and its online (*cloud*) infrastructure. Cloud infrastructure has become essential to manage, notify and update the software running on a device [18]. The characteristics of a device (e.g., operating system (OS), installed applications, and user behaviour) can therefore often be revealed through only monitoring its affiliated cloud infrastructure [19], [20].

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

[1]Henceforth referred to as *situational awareness*.

[2]For example, p0f requires the analysis of full packet captures to infer device characteristics [15]. Access to such packet captures in an operational network would be heavily restricted due to privacy and security concerns.

A bipartite graph was used to represent the affiliations between the devices on an enterprise network and the services that the devices communicate with over the Internet (Fig. 1). Bipartite graphs are comprised of two distinct sets of vertices; henceforth referred to as the *actor* (A) and *community* (C) sets. The edges of a bipartite graph are constrained such that an edge must connect an actor ($a \in A$) to a community ($c \in C$). A bipartite graph thus provides an intuitive representation between the devices on an enterprise network (*actors*) and the services (*communities*) in which they affiliate with over the Internet.

The principal challenges in analysing a bipartite graph for the purposes of device management are as follows:

1) **Network size** [21]: The sheer number and diversity of devices—and their affiliated Internet services—require the analysis of a significantly large bipartite graph.
2) **Network volatility** [21]: A bipartite graph modelled from network traffic would be highly dynamic as devices will join and leave the network, update their software, and services will continuously modify their cloud infrastructure.
3) **Aggregated cloud infrastructure** [12]: The cloud infrastructure used by distinct software may not be distinct itself. Distinct software may utilise the same cloud infrastructure if produced by the same company or operate from the same content delivery network (CDN). For example, Akamai—one of the largest CDN providers—hosts services for thousands of diverse companies including: Apple, Microsoft, Adobe, and Tiktok[3].
4) **User behaviour** [11]: The user behaviour on a device can act to obfuscate the underlying characteristics of the device itself. This may be done unintentionally; or intentionally for privacy concerns or for malicious intent.

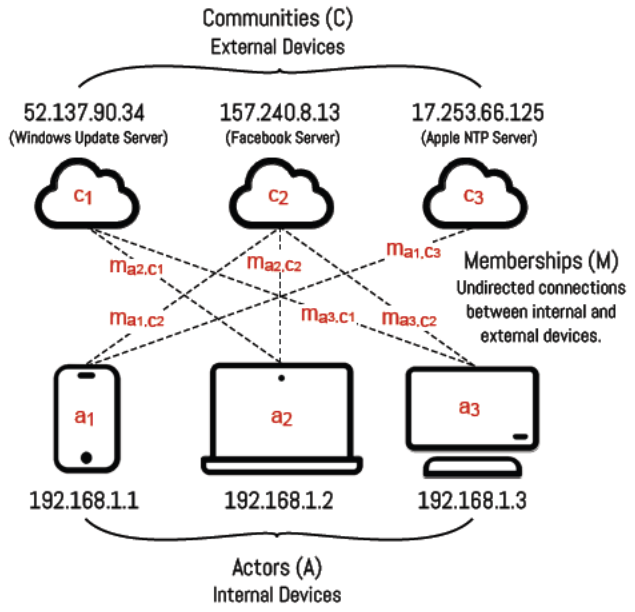


Fig. 1. An example bipartite graph created from network data. The actors and communities of the bipartite graph are the devices internal and external to the network under analysis, respectively. The membership set consists of the network communications between the internal and external devices [19].

[3]The listed companies were shown to utilise Akamai's CDN infrastructure during the capture taken of the enterprise network evaluated within this work (Section IV).

To address these challenges, we propose the use of graph embeddings to analyse the bipartite graph structure.

Graph embeddings learn to represent a graph's vertices, V, in a d-dimensional latent space, $\vec{X} \in \mathbb{R}^{|V| \times d}$. This latent space quantifies the graph's structural information into a representation that can then be easily analysed through traditional machine learning (ML) techniques.

The application of graph embeddings have been widely studied to simplify, visualise, and investigate complex networks across many distinct domains [22], [23]. We pose the use of graph embeddings to overcome the complexity of analysing the communication patterns of an enterprise network. Furthermore, graph embeddings were investigated as the embeddings they produce do not require retraining for different applications. This property would allow the same embeddings to be used for a variety of device management solutions with minimal additional computing cost.

The majority of graph embedding techniques are designed for the analysis of static, homogeneous graphs [24]–[27]. In this paper, we identify severe limitations of such techniques when applied for the analysis of an enterprise network. To address these limitations, we design and evaluate bipartite graph embeddings (BGE); the first graph embedding technique designed for the analysis of large-scale, dynamic enterprise networks. In particular, the key contribution of this work is four-fold:

1) We propose the first use of graph embeddings to provide insight into key areas of device management. That is, device characterisation, service prediction, and network visualisation. We demonstrate that the state-of-the-art graph embeddings techniques are computational restrictive when applied to large enterprise networks.
2) We design bipartite graph embeddings (BGE). BGE is the first graph embedding technique that enables the real-time analysis of a large enterprise network. We show that BGE remains effective under partial network observation and efficiently scales for the analysis of networks containing hundreds of thousands of devices.
3) We provide a novel sampling strategy—transient edge sampling (TES)—to enable BGE for the analysis of large-scale and highly-dynamic graphs. We show that TES provides an order-of-magnitude reduction in the training time of BGE when compared to current sampling strategies (i.e., edge sampling and random walks). We validate that the reduction in training time does not reduce the fidelity of the resultant graph embeddings.
4) We provide BGE as packaged tool[4] that can be used to generate insight into any bipartite graph structure (e.g., recommendation and citation networks).

The remainder of this paper is structured as follows: A survey of related work in device management and graph embeddings is provided in Section II. Section III introduces bipartite graph embeddings (BGE). Section IV outlines the dataset used for our analysis. Section V provides an empirical

[4]https://github.com/MillarK-UofA/bipartite_graph_embeddings

evaluation of the efficacy of BGE. The paper concludes in Section VI with a discussion of results and suggestions for future work.

II. Related Work / Background

A. Device Management

In recent years, machine learning (ML) has become the catalyst for developing novel network analysis solutions [10], [28]. In device management, the principal benefit of ML is its ability to automatically generate a device's fingerprint based on the device's network behaviour; thus removing the labour intensive process of manually managing a fingerprint database.

Current ML-based solutions for device management perform either deep packet inspection (DPI) or flow-based analysis:

1) **Deep packet inspection** solutions [29]–[33] exploit the idiosyncrasies in a device's packet creation process to identify the characteristics of the device.
2) **Flow-based analysis** solutions [6], [34]–[36] characterises a device based on its observed flow statistics; where, flow statistics (e.g., NetFlow [37]) are the derived metadata from an exchange of packets between two end hosts.

The key assumption of DPI is the availability of unencrypted packet content. In recent years, the validity of this assumption has been greatly diminished due to widespread encryption standards [38]. Furthermore, access to packet content is often restricted to uphold the security and privacy of the network's users [39]. Flow-based analysis has therefore become critical to maintaining effective device management solutions within modern TCP/IP networks.

A common limitation in flow-based analysis is the reliance on bespoke flow statistics. That is, flow statistics that are unlikely to be monitored across diverse TCP/IP networks (e.g., total packets with a RST flag [35] or maximum effective window size [34]). This limitation would heavily restrict the widespread application of a device management solution.

In this paper, we focus solely on device management solutions that would be widely applicable across diverse TCP/IP networks. A solution was deemed to be widely applicable if it utilises only the following nine flow statistics: the five-tuple[5], number of bytes in the flow, number of packets in the flow, and the timestamp of the flow's first and last packet. These nine flow statistics are the smallest set of features that are required to adequately describe a flow [40]; and would therefore be commonly available across diverse TCP/IP networks.

Only two device management solutions were found to be widely applicable across diverse TCP/IP networks:

1) Arora et al. [6] propose a flow-based solution for device management utilising the set of commonly available flow statistics and the flow's direction. The authors train an SVM classifier to determine whether a device is acting as server or user on an enterprise network.
2) Meidan et al. [36] propose a flow-based solution for device management utilising the set of commonly available flow statistics and the flow's TCP flags (e.g., SYN/ACK). The authors train a light gradient boosting machine (LGBM) to classify distinct types of IoT devices.

In this paper, we provide the following extensions to the two surveyed solutions for device management across diverse TCP/IP networks:

1) **Minimal Feature Set:** Both surveyed solutions utilised the set of commonly available flow statistics and an additional feature (i.e., flow direction [6] and TCP flags [36]). The additional features used in the surveyed solutions are commonly found when monitoring a TCP/IP network, however, they are not required to adequately describe a flow [40]. The additional features used by the surveyed solutions may therefore restrict their application on certain TCP/IP networks. In contrast, the solution provided within this paper utilises only the source and destination IP addresses; two fundamental features for describing a flow.
2) **Cross-Domain Applicability:** Both surveyed solutions provide a solution for a single type of device management (i.e., server/user identification [6] and device type identification [35]). In contrast, the solution provided in this paper is shown to have cross-domain applicability. That is, OS classification, service prediction, and network visualisation.
3) **Long-Term Deployment:** Both surveyed solutions utilised a supervised ML algorithm (i.e., SVM [6] and LGBM [36]). Supervised ML learning would require a large quantity of labelled network data to be updated in response to changing network behaviour. Network behaviour is highly volatile (Section I); the surveyed solutions would thus continuously require labelled network data to remain effective for long-term deployment. In contrast, the solution provided in this paper utilises unsupervised ML. In Section V-A, we show that our proposed solution can achieve an OS classification accuracy of 90% with only two labelled devices per OS (1.6% of the overall network).

To achieve these three extensions we provide a novel graph embedding technique—bipartite graph embeddings (BGE). In previous works, we have shown the effectiveness of a bipartite graph representation for device management through the analysis of IP addresses alone [19], [20]. In this work, we extended our analysis through the use of graph embeddings. We show that BGE can be applied for the analysis of large-scale, dynamic graphs such as those encountered when modelling an enterprise network. This extension is therefore essential for the real-world applicability of the proposed solution.

B. Graph Embeddings

This section provides an introduction to graph embeddings. Readers familiar with graph embeddings can skip to Section III which defines the novel graph embedding technique that is provided within this paper.

[5]The 5-tuple is a set of flow statistics composed of the source and destination IP address, source and destination port number, and the transport protocol (e.g., TCP/UDP).

Graph embeddings aim to preserve a graph's structural information within a d-dimensional latent space (1) [41]. This latent space can then be used visualisation or to perform subsequent analyses over the graph.

$$\theta : v \rightarrow \vec{x} \in \mathbb{R}^d \text{ for all } v \in V \tag{1}$$

where θ is a structure preserving mapping function from vertex, v, to the vertex's latent space representation, $\vec{x}$; d is the dimensionality of the latent space; and V is the set of all vertices within the graph.

Inspired by Mikolov *et al.* [42], recent approaches to graph embeddings have typically utilised a common optimisation strategy [24]–[27]. This optimisation strategy states that an optimal θ is that which maximises the probability of observing the neighbourhood[6] of a vertex given the vertex itself. The optimisation strategy is therefore defined as

$$\arg\max_{\theta} \sum_{v \in V} \text{logPr}(\Gamma(v)|v;\theta) \tag{2}$$

where $\Gamma(v)$ is the neighbourhood of the vertex, v, and Pr is the probability of observing $\Gamma(v)$ given v and θ.

To maximise (2), the mapping function, θ, must learn to encode which vertices belong in similar neighbourhoods. It is assumed that the probability of observing the vertices within the neighbourhood of a vertex, v, is independent given v. The assumption of conditional independence for the neighbourhood of a vertex is necessary for the defined optimisation problem to be made tractable [25]. This assumption allows (2) to be rewritten as

$$\arg\max_{\theta} \sum_{v \in V} \sum_{u \in \Gamma(v)} \text{logPr}(u|v;\theta) \tag{3}$$

Softmax and negative sampling [43] have both been investigated as optimisation strategies for (3). Negative sampling is however more commonly utilised as it has been proven to be more effective for larger scale evaluations [43]. The objective of negative sampling is to distinguish the vertices within the neighbourhood of a vertex (*positive samples*) from a set of vertices randomly sampled from the entire graph (*negative samples*). The likelihood that a sampled vertex is a positive sample as opposed to a negative sample provides an estimate of the probability component of (3).

Negative sampling is evaluated using a series of logistic regressions for both the positive and negative samples (4). These logistic regressions are then optimised using a standard feed-forward neural network and stochastic gradient descent (SGD) [44].

$$\begin{aligned} \text{Pr}(u|v;\theta) = {} & \sigma(\theta(u) \cdot \theta(v)) \\ & + \sum_{i=1}^{k} w_i \sim P_n(w)[\sigma(-\theta(w_i) \cdot \theta(v))] \end{aligned} \tag{4}$$

where σ is the sigmoid function; $\theta(\cdot)$ is the latent space representation of a vertex; k is the number of negative samples to use per positive sample; and w_i is a randomly sampled vertex from a predefined noise distribution, $P_n(w)$. The noise distribution is chosen heuristically; however, it is typically set as $P_n(w) \propto (d_w)^{\frac{3}{4}}$ as first proposed in [43]; where d_w is the degree of the vertex w.

[6]The neighbourhood of a vertex, $\Gamma(v)$, is the set of all vertices that have an edge with v.

Positive samples are selected through either random walks [24], [25], [27] or edge sampling [26]. Random walks are the preferred sampling strategy as they inherently encode the graph's local structural properties within the positive samples selected [24]. Random walks however are expensive to compute. Generating positive samples through random walks has been shown to take longer than the optimisation itself [25].

Edge sampling is an alternative method for selecting positive samples. Edge sampling generates positive samples through randomly selecting edges from the graph. The likelihood of sampling a particular edge can either be uniformly distributed or influenced by a specific attribute of the edges themselves.

The typical neural architecture used to embed the vertices of a graph is shown in Fig. 2 b). It is seen that each vertex in the graph is present at the input and output layer of a fully-connected neural network. A single hidden layer provides the representation of the input and output layers within a d-dimensional latent space ($d \ll |V|$).

The neural architecture is trained using a series of logistic regressions. Each logistic regression denotes whether a target vertex, u, in the output layer is within the neighbourhood of a source vertex, v, in the input layer. The optimal result is achieved when the logistic regression of any two vertices produces the result

$$\sigma(\theta(v) \cdot \theta(u)) = \begin{cases} 1, & \text{if } u \in \Gamma(v) \\ 0, & \text{else} \end{cases} \tag{5}$$

Each vertex in a homogeneous graph is encoded as two embeddings; once as a source vertex (input layer) and once as a target vertex (output layer). This representation is necessary as an edge may exist between any two vertices within a homogeneous graph. After training, it is common to represent a vertex by its source embedding and discard its target embedding. The structural information contained in the target embeddings is therefore lost when utilising common graph embedding techniques (e.g., node2vec [25]).

III. Bipartite Graph Embeddings

Previous graph embedding methods have shown great application for encoding the structure of static, homogeneous graphs. However, there has been limited insight in modifying this methodology to exploit the unique properties of a dynamic bipartite graph. Furthermore, the application of graph embeddings for device management has not yet been explored. In this paper, we provide bipartite graph embeddings (BGE) to address these limitations.

A. Neural Architecture

The fundamental property of a bipartite graph is that edges can only exist between vertices of the opposing set. This

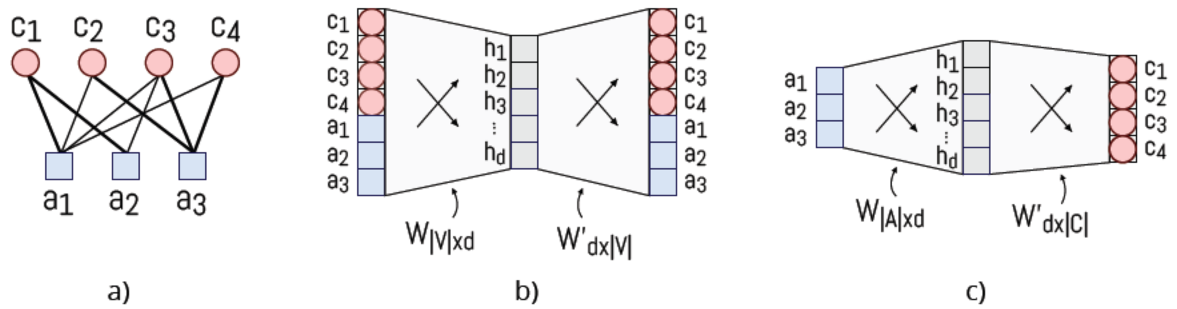


Fig. 2. The application of graph embeddings on a bipartite graph. (a) the input bipartite graph. (b) a typical neural architecture for generating graph embeddings on a homogeneous graph. (c) The neural architecture of bipartite graph embeddings (BGE).

property allows for a simplification to be made to the typical neural architecture used in graph embeddings. The neural architecture designed for BGE is shown in Fig. 2 c). It is seen that as the neighbourhood of a vertex must be of the opposing set, the vertices of the actor and community sets can be divided over the input and output layers in the neural architecture. This simplified neural architecture generates only a single embedding for each vertex in the graph.

There are four key benefits of BGE's neural architecture.

1) The structural properties of a vertex are fully encoded into a single embedding. This benefit ensures that the structural information of a vertex is not lost through its separation into source and target embeddings.
2) The embeddings generated through BGE's architecture are directly optimised for link prediction between the actor and community sets. In contrast, the typical neural architecture is optimised for link prediction between the source and target embeddings. The optimisation of the typical architecture is therefore sub-optimal for link prediction as only the source embeddings are retained for evaluation.
3) BGE's architecture ensures that only feasible edges in a bipartite graph are used in training. In contrast, the typical neural architecture will learn a relationship between any two vertices; thus vastly increasing the complexity of the model without generating further insight.
4) The size of BGE's architecture is reduced by half leading to lower memory and storage requirements of BGE and enabling its use for the analysis of larger graphs.

B. Transient Edge Sampling

To update (4), $k+1$ edges must be selected—one positive sample and k negative samples. Conventionally, random walks or edge sampling are utilised to select positive samples; whereas, negative samples are drawn randomly from a pre-defined noise distribution, $P_n(w)$. Selecting both positive and negative samples require a pre-sampling stage where the prior distributions of the graph must be known. An example of the positive and negative samples generated from conventional edge sampling is shown in Fig. 3 a). It is seen that a single edge (v, u) is selected as the positive sample; whereas, k negative samples, $(w_i)_{i=1}^{k}$, are randomly sampled from the underlying graph.

An alias table can be used to efficiently select positive and negative samples from a prior distribution [26]. An alias table, however, assumes a static prior distribution and thus is sub-optimal for the analysis of dynamic graphs such as those encountered when modeling an enterprise network. Furthermore, the computational complexity of generating a graph's prior distributions would be impractical at the scale necessary to analyse large enterprise networks. To address these limitations, we devise a new sampling strategy—transient edge sampling (TES).

The aim of TES is to generate positive and negative samples without sampling from a distribution of the underlying graph. Achieving this aim eliminates the most severe computational restrictions of applying graph embeddings to a large-scale dynamic graph and thus allow for the practical application of graph embeddings for network analysis. Additionally, TES greatly reduces the evaluation time when applied to a static graph by removing the need to generate prior distributions (Section V-D).

TES removes the reliance on a graph's underlying distribution by exploiting the continuous stream of traffic generated by enterprise networks. Positive samples are generated for each flow in the network; whereas, negative samples are drawn

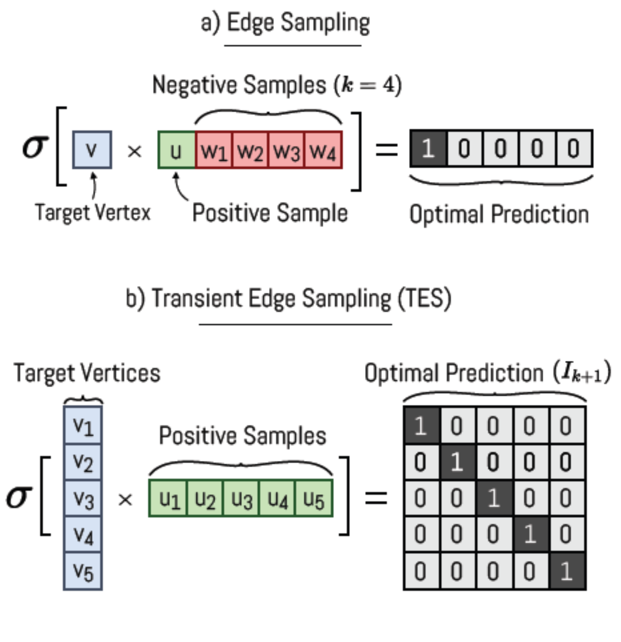


Fig. 3. An illustration of the samples selected using a) conventional edge sampling and b) transient edge sampling.

TABLE I
BIPARTITE GRAPHS STATISTICS (1^{st} MAY 2019); WHERE, $|A|$ IS THE NUMBER OF DEVICES, $\mu_{deg(A)}$ IS THE MEAN NUMBER OF COMMUNITIES PER DEVICE, $|C|$ IS THE NUMBER OF COMMUNITIES, $\mu_{deg(C)}$ IS THE MEAN NUMBER OF DEVICES IN EACH COMMUNITY, AND $|M|$ IS THE NUMBER OF UNIQUE ACTOR-COMMUNITY AFFILIATIONS.

Bipartite Graph	Community Definition	$\|A\|$	$\mu_{deg(A)}$	$\|C\|$	$\mu_{deg(C)}$	$\|M\|$
bipartite-IP	IP	498	205.0	19.2k	5.3	102.1k
bipartite-ClassC	Class C		120.7	12.3k	4.9	60.1k
bipartite-WHOIS	WHOIS		89.6	3.7k	12.1	44.6k

TABLE II
THE DISTRIBUTION OF OPERATING SYSTEMS ON THE UNIVERSITY'S WIRELESS NETWORK ON THE 1^{st} OF MAY 2019.

Operating System	Actors	
iOS	240	(48%)
Windows	45	(9%)
OS X	92	(19%)
Android	121	(24%)
Total	498	(100%)

from subsequent flows. As shown in Fig. 3 b), TES draws $k+1$ edges to select both the positive and negative samples. A matrix multiplication is then applied over the embeddings of the selected vertices and the element-wise logistic regression is applied over the resultant matrix. This operation produces a prediction, $\hat{Y} \in \mathbb{R}^{(k+1)\times(k+1)}$. The optimal prediction is achieved when $\hat{Y}$ is equal to the identity matrix, I_{k+1}.

A limitation of TES is that it assumes that there is diversity in the transient edges sampled. Repeated vertices in the sampled edges will cause sub-optimal updates to be made when optimising (4). The likelihood of sampling repeated vertices is dependent on the number of vertices within the graph and the relative activity of each vertex. While this requirement may exclude the utilisation of TES in some analyses, we show that TES is well-suited for the analysis of an enterprise network (Section V).

The benefit of TES is that only the transient edges of a graph are required to select both positive and negative samples. This allows TES to be 1) computationally efficient, as training samples are not derived from the analysis of the entire graph; and, 2) enables the analysis of highly dynamic graphs for which experience rapid influxes of new edges (e.g., enterprise networks).

IV. Dataset

The evaluation of BGE was performed on a university wireless network. The network under analysis allows for both staff and students alike to connect their wireless BYO devices into the university's enterprise network. The university's wireless network provides our analysis with real-word network conditions for which to evaluate the efficacy of BGE.

A 26-hour capture was taken on the university's wireless network over a typical workday. The statistics of a bipartite graph constructed from the full capture is shown in Table I. The actor and community sets in the constructed bipartite graph are the devices internal and external to the university's enterprise network, respectively.

The actors under analysis were chosen for the evaluation of operating system (OS) classification (Section V-A). Actors running iOS, Windows, OS X, and Android OSs were selected as their combined market share represents over 96% of total device usage worldwide [45]. The proportion of devices running each evaluated OS is shown in Table II.

The communities under analysis were not restricted by any heuristic. Each community, however, was defined by its IP address, Class C subnet, and WHOIS subnet. The WHOIS subnet of each community was resolved through an API request to DNSlytics[7]. Each community definition produced a distinct bipartite graph representation as shown in Table I. The community definitions (IP, Class C, and WHOIS) were evaluated in Section V-B to determine their efficacy for OS classification.

V. Results

We evaluate BGE on two tasks specific to device management; OS classification (Section V-A) and service prediction (Section V-C). For comparison, we evaluate BGE against three state-of-the-art graph embedding techniques:

1) Node2vec [25]: A homogeneous graph embedding technique that utilises a biased random walk to generate positive samples. Graphs embeddings are trained through applying the word2vec algorithm [42] over the corpus generated through random walks.
2) LINE [26]: A homogeneous graph embedding technique that utilises edge sampling to generate positive samples. The embeddings generated by LINE consist of two components; representing the first- and second-order relationships between vertices, respectively. A vertex's embedding is defined by concatenating its first- and second-order components.
3) Metapath2vec++ [27]: A heterogeneous graph embedding technique that utilises a meta-path random walk to generate positive samples. Graph embeddings are trained using a heterogeneous variation of the word2vec algorithm.

We exclude a bipartite graph embedding approach, BiNE [46], as we were unable to produce meaningful results with their provided technique[8] on the dataset under analysis.

The hyperparameters used for each technique were made equivalent where possible. In particular, we set $d = 128$, $r = 10$, $l = 80$, $s = 10$, $k = 5$ as proposed in [25]; where d is the dimensionality of the embeddings, r is the number of random walks per vertex, l is the length of each random walk, s is the size of word2vec's context window, and k is the number of negative samples. To train each technique, we generate an equal number of training samples as proposed in [25]; where the sampling budget is defined as $\kappa = r \times l \times |V|$.

[7]https://dnslytics.com/
[8]https://github.com/clhchtcjj/BiNE

A. *OS Classification*

The aim of OS classification is to identify the type of OS running on a device. Identifying a device's OS is a crucial step when performing device management. It first allows for the identification of major security vulnerabilities that are tied to the device's OS. For example, the infamous WannaCry ransomware attack targeted the EternalBlue exploit within Windows OSs [47]. Second, OS classification facilitates the management of end-point devices as distinct OSs often require distinct managerial strategies. For example, a network that is composed predominately of devices running the OS X and iOS OSs can managed through the deployment of a JAMF server [48].

A significant challenge in OS classification is the rate in which OSs are updated [20]. Each update has the potential to change the underlying network behaviour of an OS and thus degrade the classifier's performance. OS classification techniques must therefore account for this volatility as it is impractical to retrain a new classifier whenever OS classification is performed.

Graph embeddings are an unsupervised ML algorithm and therefore do not require labelled data. In theory, graph embeddings could therefore be continuously trained such that a device's embedding evolves with the changing behaviour of the device itself. However, typical graph embeddings techniques cannot be applied to dynamic graphs and thus cannot be applied for such applications. In contrast, the TES sampling strategy enables the use of graph embeddings to evaluate the evolving characteristics of an OS through the analysis of a dynamic graph—mitigating the effect of OS volatility.

Graph embedding techniques are unsupervised and therefore can be used in a variety of applications such as visualisation or to detect unknown or anomalous OSs. Fig. 4 provides a visualisation of the device embeddings produced BGE. Distinct OS clusters are observed to form through BGE's unsupervised training. A small portion of labelled data would be required, however, to classify the distinct OSs that these clusters represent. Fortunately, within an enterprise network there is likely to be a set of managed devices for which their OS is known *a priori*. This set of known devices (referred to as *seed devices* in this work) serve as an example of the typical behaviour of OSs at the time of analysis. These seed devices can be used to label the OS clusters produced by BGE.

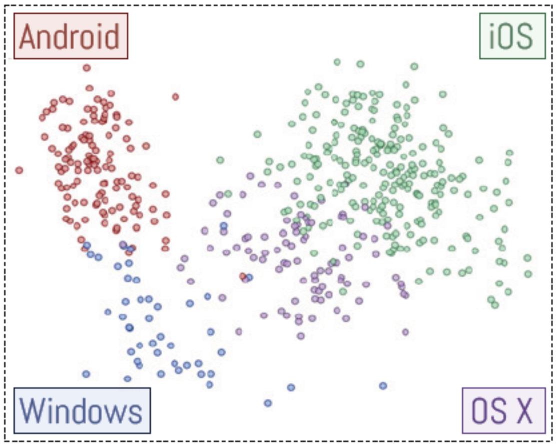


Fig. 4. A visualisation of the device embeddings produced by BGE on the biparitite-IP graph. The embeddings were trained in 128 dimensions. PCA was used to project the embeddings into a 2-dimensional space for visualisation purposes. The device embeddings are coloured based on their respective OS: iOS (green), OS X (purple), Windows (blue), and Android (red).

To perform OS classification, we evaluate k-means clustering [49] on the device embeddings generated through BGE. We initialise k-means clustering using a set of seed devices as the initial centroids for each OS cluster. The classification performance was then evaluated over a varying number of seed devices per OS to identify the required number of known devices to accurately predict the OS composition of a network.

Table III depicts the OS classification performance of k-means clustering on the embeddings generated through each of the evaluated techniques. All embedding techniques were shown to facilitate accurate predictions using ten or fewer seed devices per OS. That is, over a 90% classification accuracy was achieved when less than 8% of devices on the network are known. Graph embeddings can therefore provide an effective strategy to classify the OS of the devices on a network that requires only the analysis of the IP address field and minimal labelled devices.

The performance of BGE when using conventional edge sampling (ES) and random walks (RW) was also investigated in Table III. Conventional edge sampling and random walks were implemented based on the algorithms used in LINE [26] and node2vec [25], respectively. Overall, TES showed a slight improvement in accuracy when compared with both conventional sampling strategies as well as the compared state-of-the-art graph embedding techniques. This result verifies that the removal of the edge sampling stage does not reduce the structural information contained within the graph embeddings produced by TES. In Section V-D, we show that this removal of the edge sampling stage provides an order-of-magnitude reduction in the training time of BGE when compared to the conventional sampling strategies. TES thus provides a sampling strategy that drastically reduces the training time of BGE without impacting the overall quality of the resultant embeddings[9].

B. *Community Aggregation*

Communities thus far have been defined by their respective IP address. Distinct community definitions could however be applied to encode alternative community structures or reduce the number of communities to be analysed. For example, aggregating communities that provide similar services would 1) reduce the overall size of the network and 2) produce higher quality embeddings by explicitly grouping communities that exhibit similar behaviour.

In this section, we investigate the performance of BGE's OS classification performance in response to three community definitions; the server's 1) IP address, 2) Class C subnet, and 3) WHOIS subnet. Aggregating communities by their respective

[9]For the remainder of the paper, all evaluations of BGE will utilise the TES sampling strategy unless otherwise stated.

TABLE III
OS CLASSIFICATION PERFORMANCE OF BGE ON THE BIPARTITE-IP GRAPH. K-MEANS CLUSTERING WAS USED TO CLASSIFY THE EMBEDDINGS GENERATED BY BGE. RESULTS DEPICT THE MEAN MACRO F1-SCORE OVER 10 TRIALS. FOR COMPARISON, THE PERFORMANCE OF NODE2VEC, LINE, AND METAPATH2VEC++ IS PROVIDED. BGE WAS EVALUATED USING THREE EDGE SAMPLING TECHNIQUES: TRANSIENT EDGE SAMPLING (TES), CONVENTIONAL EDGE SAMPLING (ES) AND RANDOM WALKS (RW).

Technique	# Seed Devices									
	1	2	3	4	5	6	7	8	9	10
node2vec	0.762	0.883	0.848	0.873	0.897	0.929	0.922	0.922	0.925	0.943
LINE	0.792	0.849	0.893	0.903	0.928	0.928	0.929	0.936	0.934	0.937
metapath2vec++	0.573	0.823	0.755	0.860	0.772	0.873	0.874	0.883	0.886	0.890
BGE-TES	**0.824**	**0.899**	**0.926**	0.909	**0.936**	**0.931**	**0.949**	**0.946**	**0.944**	**0.946**
BGE-ES	0.808	0.822	0.891	**0.915**	0.895	0.901	0.927	0.926	0.923	0.926
BGE-RW	0.755	0.805	0.884	0.901	0.904	0.907	0.913	0.922	0.923	0.931

subnet substantially reduces the size of the resultant graph—as shown in Table I—whilst introducing negligible overhead to the graph's construction. Subnet aggregation, however, will incur a loss of information as distinct services can be represented by the same community.

Fig. 5 compares the performance of subnet aggregation for OS classification. Aggregating communities by their subnet was shown to significantly reduce BGE's OS classification performance. This result confirms that distinct OS systems are likely to affiliate with services hosted within the same subnets. It is therefore inadvisable to aggregate communities by their subnet for OS classification.

Community aggregation strategies could be defined to optimise BGE for different tasks. For example, aggregating communities by their domain is likely to be beneficial for tasks such as identifying the applications installed on a device. Domain aggregation, however, would incur significant overhead as it would require the domain of an IP address to be resolved. An in-depth investigation of alternate community definitions is left subject for future work. For the remainder of the paper, all evaluations will define communities by their associated IP address.

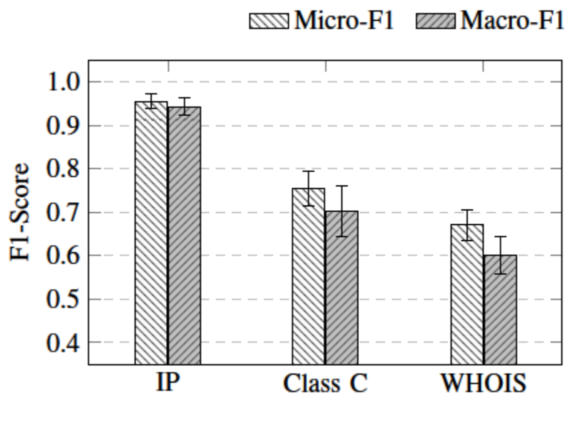


Fig. 5. OS classification performance of distinct community aggregations. Communities were aggregated by their IP address, Class C subnet and WHOIS subnet, respectively. Results were obtained from ten trials of BGE using 10 seed devices per OS for the initialisation of k-means clustering. The mean and standard deviation of the Micro and Macro F1-Scores of the ten trials are depicted.

C. Service Prediction

The forecasting of future traffic demand is essential for providing a higher quality-of-service on a network [50], [51]. One component of traffic forecasting is service prediction. The aim of service prediction is to predict which services a device is likely to use. Service prediction thus allows a network operator to pre-allocate appropriate network resources to better manage the devices on a network [12]. In this work, link prediction was investigated as a method of service prediction through estimating the likelihood that a device will communicate with a particular service on the Internet.

To evaluate link prediction, 50% of the edges were randomly removed from the bipartite-ip graph. Edges were removed such that no vertex would be disconnected from the resultant graph. Embeddings were then trained on the resultant graph using the evaluated embedding techniques. The removed edges were used within the testing set as examples of true edges; whereas, false edges were generated by sampling an equal number of edges from random vertex pairs.

Logistic regression was used to identify the likelihood that an edge exists between an actor and a community. To evaluate logistic regression, edges were represented by the aggregation of their respective actor and community embeddings. Four aggregation operations were evaluated to assess their performance on link prediction:

$$\sigma(\theta(a) \circ \theta(c)) = \begin{cases} 1, & \text{if link exists} \\ 0, & \text{else} \end{cases} \tag{6}$$

a) $\circ_{cos} = \frac{\theta(a) \cdot \theta(c)}{||\theta(a)|| \times ||\theta(c)||}$ b) $\circ_{had} = \theta_i(a) \times \theta_i(c)$

c) $\circ_{L1} = |\theta_i(a) - \theta_i(c)|$ d) $\circ_{L2} = |\theta_i(a) - \theta_i(c)|^2$

where $\theta(a)$ and $\theta(c)$ are the embeddings for the actor and community vertex, respectively; σ is the sigmoid function; and $\circ$ is the aggregation operation. The evaluated aggregation operations were Salton's cosine similarity, $\circ_{cos}$; the Hadamard product, $\circ_{had}$; weighted L1, $\circ_{L1}$; and weighted L2, $\circ_{L2}$. The

TABLE IV
THE SERVICE PREDICTION PERFORMANCE OF BGE USING DISTINCT AGGREGATION OPERATIONS. RESULTS DEPICT THE MEAN AREA-UNDER-CURVE (AUC) SCORES OF TEN TRAILS.

Technique	Cosine	Hadamard	Weighted L1	Weighted L2
node2vec	0.578	0.660	0.773	0.782
LINE	0.720	**0.801**	0.785	0.788
metapath2vec++	0.570	0.757	**0.813**	**0.819**
BGE	**0.838**	0.631	0.807	0.816

definitions presented show the ith component of the element-wise aggregation operations.

Table IV reports the link prediction performance of each embedding technique when utilising the four aggregation operations. The highest reported performance was observed with BGE-Cosine. BGE's performance increase when using the cosine aggregation operation is due to the design of BGE's neural architecture. BGE is the only evaluated technique that is directly optimised through the dot product of the actor and community embeddings. This property is exploited when using the cosine (i.e., a normalised dot product) aggregation operation when performing link prediction.

The evaluated graph embedding techniques all report similar link prediction performance when using one of the four aggregation operations. Overall, BGE-Cosine is shown to be the most effective graph embedding technique for link prediction due to its scalability (Section V-D) and slight improvement in link prediction performance.

D. Scalability

A set of random bipartite graphs were generated to investigate the scability of TES. Each bipartite graph was generated such that the average degree of the actor and community set was consistent with the bipartite-ip graph (Table I). The number of actor vertices was increased from 10 to 100,000 to investigate the time required to evaluate different sized enterprise networks. BGE was evaluated using the TES sampling strategy as well as random walks (RW) [25] and conventional edge sampling (ES) [26].

Fig. 6 illustrates the time required to evaluate each of the sampling strategies on various sized enterprise networks. This result showcases the clear advantage of the TES sampling strategy. Overall, BGE-TES was shown to be order-of-magnitude faster than both BGE-RW and BGE-ES. Furthermore, BGE-TES was shown to be at least four times faster than the sampling time alone of both RW and ES. The application of TES would therefore be well suited for the analysis of static graphs to greatly improve the evaluation time.

The results of Fig. 6 showcases the applicability of TES for the analysis of large-scale enterprise networks. TES was estimated to take less than four hours to evaluate an enterprise network consisting of a hundred thousand devices[10]. In contrast, the use of conventional sampling techniques that are used in the state-of-the-art graph embedding techniques would result in an evaluation time of over 33 hours. Furthermore, conventional sampling techniques would require the graph to be re-sampled to evaluate new flows in the network. This requirement would drastically reduce the efficacy of conventional graph embedding techniques for the analysis of an enterprise network.

[10]This experiment assumed a 26-hour capture period for which approximately four million distinct community services were contacted.

E. Network Fidelity

A device's observed network behaviour may not be completely indicative of the device's true characteristics. Devices may join and leave the network, utilise cellular data for certain services, or user activity on the device could obfuscate the device's underlying characteristics. This behaviour would reduce the fidelity of the network representation analysed through BGE. It is therefore important to investigate the effect of network fidelity on BGE's performance.

Fig. 7 shows the effect of network fidelity on BGE's OS classification performance on the bipartite-ip graph. The fidelity of the graph was reduced through modifying its edges. Edges were either randomly added to or removed from the graph:

1) **Added Edges [Noisy observations]**: Edges were randomly added to the bipartite-ip graph to simulate noisy observations of network traffic. Noisy observations may occur unintentionally, when there is erratic and diverse user activity on the network; or intentionally, when a malicious actor deliberately introduces noisy observations to obfuscate analysis. It is seen that the performance of BGE drops by only 5.3% when 50% of the evaluated graph is comprised of random edges. BGE therefore proves to be a suitable option for fostering situational awareness within noisy network environments.
2) **Removed Edges [Missing Observations]**: Edges were randomly removed from the bipartite-ip graph to simulate missing observations of network traffic. Missing observations are likely to occur due to devices leaving and

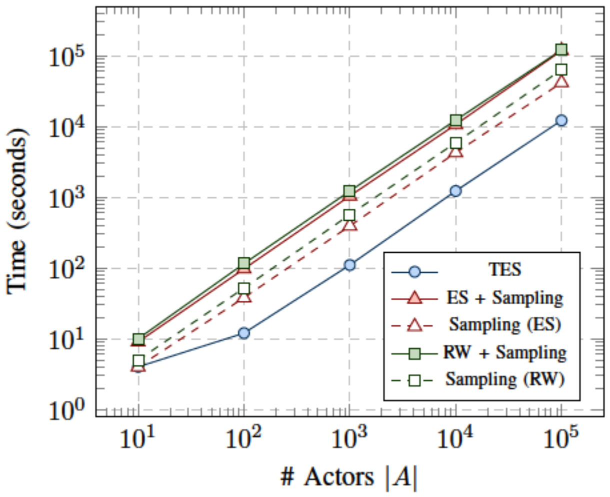


Fig. 6. Scalability of BGE when using the TES sampling strategy. For comparison, BGE was also evaluated using random walks (RW) and conventional edge sampling (ES).

rejoining the network—a common occurrence with BYO user devices. The performance of BGE was shown to drop by only 3.7% when 50% of the edges in the bipartite-ip graph were removed. The performance of BGE, however, greatly declines when less than 50% of the network is observable. BGE could therefore be reliably applied to foster situational awareness when at least 50% of the network is available for analysis.

The evaluated scenarios illustrate the robustness of BGE when the fidelity of the network observations is diminished. It was seen that BGE can be reliably applied within noisy networks environment and when only partial network observations are available. These scenarios further validate the real world applicability of BGE.

F. Hyperparameter Sensitivity

BGE has several hyperparameters that can be tuned for the analysis of a bipartite graph. These hyperparameters allow BGE to be optimised for diverse evaluations. Tuning these hyperparameters for each evaluation would, however, incur considerable overhead. The sensitivity of BGE's performance relative to its hyperparameters is therefore important to investigate its real world applicability.

The OS classification performance of BGE with respect to its principal hyperparameters is shown in Fig. 8. The hyperparameters evaluated were i) the number of negative samples (k), ii) the embedding dimensionality (d), and iii) the sampling budget (κ). These hyperparameters were chosen as they were shown to exhibit the greatest affect on BGE's OS classification performance.

The hyperparameters evaluated in Fig. 8 are shown to be relatively stable after a cut-off threshold is met. This result indicates that some experimentation may be required to tune BGE for certain network environments; however, an exhaustive search of all possible hyperparameters is unlikely to be required to generate optimal results.

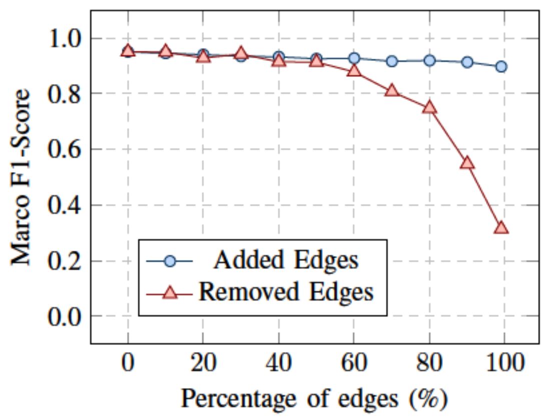


Fig. 7. Effect of network fidelity on BGE's OS classification performance. Edges were randomly added to or removed from the bipartite-ip graph to decrease its fidelity. Results were obtained from ten trials of BGE using 10 seed devices per OS for the initialisation of k-means clustering.

VI. Conclusion and Future Work

In this paper, we provided bipartite graph embeddings (BGE); the first graph embedding technique that enables the real-time analysis of a large enterprise network. The proposed technique monitors only the IP addresses that the devices affiliates with over the Internet. It is therefore encryption invariant and can be universally applied to diverse TCP/IP network configurations.

The performance of BGE was validated in three key areas of device management. That is, device characterisation, service prediction, and network visualisation. Furthermore, we showed that BGE remains effective under partial network observation and efficiently scales for the analysis of networks containing hundreds of thousands of devices

We designed a novel edge sampling strategy, transient edge sampling (TES), to evaluate highly-dynamic graphs such as those encountered when modelling large enterprise networks. We showed that TES provides an order-of-magnitude reduction in training time when compared to conventional edge sampling strategies (i.e., random walk and edge sampling). We showed that the reduction in training time does not reduce the fidelity of the resultant graph embeddings.

The performance of BGE in service prediction supports its use for predicting the applications used by a device. In future work, the use of BGE will be investigated for mobile application prediction (e.g., Facebook, Instagram, and Gmail). Mobile application prediction would provide further insight into the vulnerabilities and utilisation of a TCP/IP network.

Private networks—such as virtual private networks (VPNs)—would obfuscate the overall network behaviour of a device and thus reduce the efficacy of BGE. In future work, we will investigate an extension of BGE for the identification and characterisation of a private network.

References

[1] S. Webster, R. Lippmann, and M. Zissman, "Experience using active and passive mapping for network situational awareness," in *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, pp. 19–26, IEEE, 2006.

[2] Y. Livnat, J. Agutter, S. Moon, and S. Foresti, "Visual correlation for situational awareness," in *IEEE Symposium on Information Visualization (INFOVIS)*, pp. 95–102, 2005.

[3] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen, "Network reconnaissance," *Network Security*, vol. 2008, no. 11, pp. 12–16, 2008.

[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1093–1110, 2017.

[5] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar, "The rise of traffic classification in IoT networks: A survey," *Journal of Network and Computer Applications*, vol. 154, p. 102538, 2020.

[6] D. Arora, K. F. Li, and A. Loffler, "Big data analytics for classification of network enabled devices," in *Advanced Information Networking and Applications Workshops (WAINA)*, pp. 708–713, 2016.

[7] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[8] A. Hemmer, M. Abderrahim, R. Badonnel, J. François, and I. Chrisment, "Comparative assessment of process mining for supporting iot predictive security," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 1092–1103, 2021.

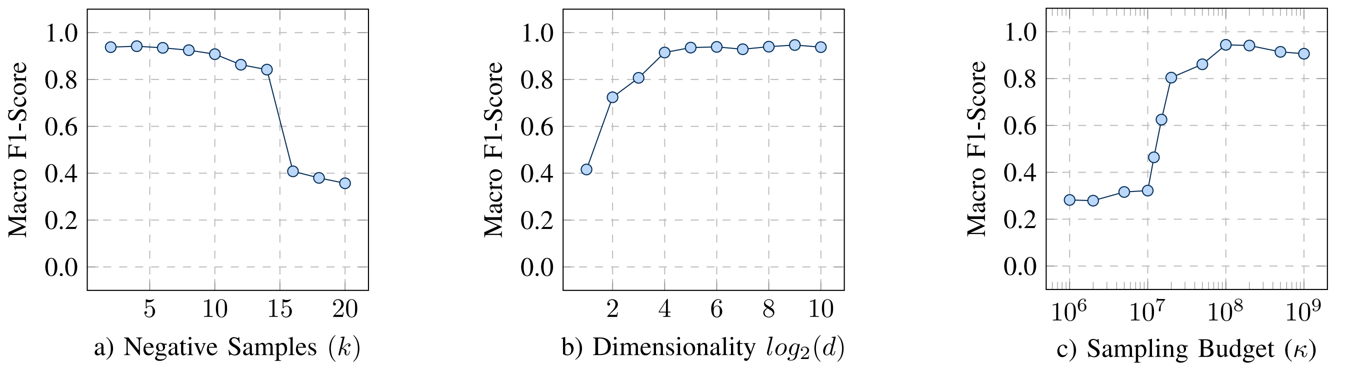


Fig. 8. OS classification performance of BGE under various hyperparameter configurations; where k is the number of negative samples per positive sample, d is the dimensionality of the embedding space, and κ is the number of positive training samples. Results were obtained from ten trials of BGE using 10 seed devices per OS for the initialisation of k-means clustering.

[9] X. Wei, N. C. Valler, H. V. Madhyastha, I. Neamtiu, and M. Faloutsos, "Characterizing the behavior of handheld devices and its implications," *Computer Networks*, vol. 114, pp. 1–12, 2017.

[10] M. Conti, Q. Li, A. Maragno, and R. Spolaor, "The dark side (-channel) of mobile devices: A survey on network traffic analysis," *arXiv preprint arXiv:1708.03766*, 2018.

[11] G. Aceto, G. Bovenzi, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Characterization and prediction of mobile-app traffic using markov modeling," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 907–925, 2021.

[12] P. A. Frangoudis, L. Yala, and A. Ksentini, "AWESoME: Big data for automatic web service management in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 13–26, 2018.

[13] G. F. Lyon, *Nmap Network Scanning: The official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[14] C. Deliang, L. Xing, and Z. Qianli, "A comparative study on user characteristics of fixed and wireless network based on DHCP," in *IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pp. 327–330, 2016.

[15] M. Zalewski, "p0fv3." lcamtuf.coredump.cx, https://lcamtuf.coredump.cx/p0f3. (accessed Mar. 3, 2020).

[16] M. Caselli, D. Hadžiosmanović, E. Zambon, and F. Kargl, "On the feasibility of device fingerprinting in industrial control systems," in *International Workshop on Critical Information Infrastructures Security*, pp. 155–166, Springer, 2013.

[17] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[18] M. Jelassi, C. Ghazel, and L. A. Saïdane, "A survey on quality of service in cloud computing," in *3rd International Conference on Frontiers of Signal Processing (ICFSP)*, pp. 63–67, 2017.

[19] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Characterising network-connected devices using affiliation graphs," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, IEEE, 2020.

[20] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Operating system classification: A minimalist approach," in *International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 143–150, IEEE, 2020.

[21] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[22] W. Nelson, M. Zitnik, B. Wang, J. Leskovec, A. Goldenberg, and R. Sharan, "To embed or not: Network embedding as a paradigm in computational biology," *Frontiers in Genetics*, vol. 10, 2019.

[23] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[24] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *20th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 701–710, 2013.

[25] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *22nd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 855–864, 2016.

[26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *24th International Conference on World Wide Web*, pp. 1067–1077, 2015.

[27] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *23rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pp. 135–144, 2017.

[28] Z. Li, A. L. G. Rios, and L. Trajković, "Machine learning for detecting anomalies and intrusions in communication networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2254–2264, 2021.

[29] N. Ammar, L. Noirie, and S. Tixeuil, "Autonomous identification of IoT device types based on a supervised classification," in *International Conference on Communications (ICC)*, pp. 1–6, 2020.

[30] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, "OS fingerprinting and tethering detection in mobile networks," in *Conference on Internet Measurement*, pp. 173–180, 2014.

[31] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *2017 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2017.

[32] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive OS fingerprinting methods in the jungle of wireless networks," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2018.

[33] P. Chemouil, P. Hui, W. Kellerer, Y. Li, R. Stadler, D. Tao, Y. Wen, and Y. Zhang, "Special issue on artificial intelligence and machine learning for networking and communications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1185–1191, 2019.

[34] A. S. Khatouni, L. Zhang, K. Aziz, I. Zincir, and N. Zincir-Heywood, "Exploring NAT detection and host identification using machine learning," in *15th IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–8, 2019.

[35] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *ACM Symposium on Applied Computing*, pp. 506–509, 2017.

[36] Y. Meidan, V. Sachidananda, H. Peng, R. Sagron, Y. Elovici, and A. Shabtai, "A novel approach for detecting vulnerable IoT devices connected behind a home NAT," *Computers & Security*, vol. 97, p. 101968, 2020.

[37] Cisco, "Netflow version 9 flow-record format." cisco.com, https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html. (accessed Oct. 17, 2021).

[38] European Network and Information Security Agency (ENISA), "Encrypted traffic analysis." enisa.europa.eu, https://www.enisa.europa.eu/publications/encrypted-traffic-analysis. (accessed Mar. 15, 2022).

[39] J. Kampeas, A. Cohen, and O. Gurewitz, "Traffic classification based on zero-length packets," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1049–1062, 2018.

[40] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[41] A. Caciularu, N. Raviv, T. Raviv, J. Goldberger, and Y. Be'ery, "perm2vec: Attentive graph permutation selection for decoding of error correction codes," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 79–88, 2021.

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[43] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3111–3119, 2013.

[44] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.

[45] Statcounter, "Operating system market share worldwide (March 2020 - March 2021)." gs.statcounter.com, urlhttps://gs.statcounter.com/os-market-share. (accessed Apr. 21 2021].

[46] M. Gao, L. Chen, X. He, and A. Zhou, "Bine: Bipartite network embedding," in *41st International ACM Conference on Research & Development in Information Retrieval (SIGIR)*, pp. 715–724, 2018.

[47] H. Hu, Y. Liu, C. Chen, H. Zhang, and Y. Liu, "Optimal decision making approach for cyber security defense using evolutionary game," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1683–1700, 2020.

[48] K. Dietiker, "Managing iOS mobile devices," in *39th ACM Conference on User Services (SIGUCCS)*, pp. 49–52, 2011.

[49] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100–108, 1979.

[50] D. H. L. Oliveira, T. P. d. Araujo, and R. L. Gomes, "An adaptive forecasting model for slice allocation in softwarized networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 94–103, 2021.

[51] D. Ferreira, A. B. Reis, C. Senna, and S. Sargento, "A forecasting approach to improve control and management for 5G networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1817–1831, 2021.

Chapter 6

Graph-Based Machine Learning for Private Network Analysis

In this chapter, we provide the first utilisation of graph embeddings for private network analysis. Private networks—such as a virtual private network (VPN)—have posed significant challenges for network reconnaissance as they deny direct visibility into their composition. In this chapter, we develop an extension of BGE (Chapter 5) to provide the first known solution for inferring the composition of both the devices and applications acting behind diverse private networks.

A single paper is presented within this chapter:

Paper 6.1 - "*PiPiN: Acquiring Situational Awareness Behind Private Networks with Confidence*", utilises graph embedding to infer the device and application composition within a private network. This paper proves that graph embeddings are semantically meaningful under vector combination. We exploit this property to decompose the embedding generated for a private network into a vector combination of its constituent devices and applications. Furthermore, this paper provides a metric of confidence for the model's prediction, allowing a network operator to make more informed decisions when administrating their network.

The key contribution of this chapter is as follows:

1. (**Paper 6.1**) We validate the use of prediction intervals (PIs) and ensemble models to provide a confidence metric for ML-based network reconnaissance solutions. This confidence metric addresses a severe limitation in related ML-based solutions as they provide insufficient insight into the uncertainty of their predictions. In contrast, we provide a metric to inform the network operator as to whether the predictions made can be accepted or whether they require further analysis. Additionally, this confidence metric can be utilised to identify when an ML-based solution requires retraining based on changes within the network environment (CN4).

2. **(Paper 6.1)** We provide PiPiN; an extension of BGE that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. PiPiN exploits a novel insight into the vector decomposition of graph embeddings to characterise a private network based only on its community structure (i.e., the IP address field). Its use is thus encryption invariant (CN1) and can be widely deployed across diverse TCP/IP private networks (CN2). Additionally, PiPiN preserves the privacy of a private network as it cannot be used to identify the characteristics of an individual device or user.

3. **(Paper 6.1)** PiPiN is the first technique that has been shown to be effective for large-scale private network analysis that is both privacy preserving and encryption invariant. We show that PiPiN can accurately estimate the number of devices within a private network consisting of up to 500 devices. Furthermore, we show that PiPiN can estimate the composition of device manufacturers, operating systems, and applications that are used within a private network. PiPiN therefore provides a network operator with crucial insight into the characteristics of a private network.

Statement of Authorship

Title of Paper	PiPiN: Acquiring Situational Awareness Behind Private Networks with Confidence
Publication Status	☐ Published ☐ Accepted for Publication [✓ Submitted for Publication] ☐ Unpublished and Unsubmitted work written in manuscript style
Publication Details	submitted to IEEE Transactions on Network and Service Management.

Principal Author

Name of Principal Author (Candidate)	Kyle Millar		
Contribution to the Paper	Conception, design, analysis, and interpretation of data. Drafted the manuscript and acted as corresponding author.		
Overall percentage (%)	70%		
Certification:	This paper reports on original research I conducted during the period of my Higher Degree by Research candidature and is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in this thesis. I am the primary author of this paper.		
Signature		Date	15/09/22

Co-Author Contributions

By signing the Statement of Authorship, each author certifies that:

i. the candidate's stated contribution to the publication is accurate (as detailed above);

ii. permission is granted for the candidate in include the publication in the thesis; and

iii. the sum of all co-author contributions is equal to 100% less the candidate's stated contribution.

Name of Co-Author	Adriel Cheng		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Hong Gunn Chew		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

Name of Co-Author	Cheng-Chew Lim		
Contribution to the Paper	Supervised the development of work, assisted in data interpretation, and provided critical revisions to the manuscript.		
Signature		Date	8/9/22

PiPiN: Acquiring Situational Awareness Behind Private Networks with Confidence

Kyle Millar, Adriel Cheng, Hong Gunn Chew, and Cheng-Chew Lim

***Abstract*—Private networks are ubiquitous across the Internet. They are essential for administrating small home networks to large enterprise networks and Internet service providers (ISPs). Furthermore, the use of private networks has been greatly expanded by the pervasive adoption of virtual private networks (VPNs). This adoption of private networks has introduced significant challenges for device management and network security through the decreased visibility into the composition of such networks. To counter these challenges, we pose PiPiN; a graph-based machine learning tool that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. PiPiN performs analysis on only the source and destination IP address field; making the technique encryption invariant and widely deployable across diverse TCP/IP networks.**

PiPiN was evaluated on three captures of a university campus network. We show that PiPiN can accurately estimate the number of devices within a private network consisting of up to 500 devices. Furthermore, we show that PiPiN provides comprehensive situational awareness behind a private network through estimating its composition of device manufacturers, operating systems, and applications.

***Index Terms*—IP network, machine learning, security management, network management, graph embeddings.**

I. Introduction

PRIVATE networks are a foundational component of the Internet. A private network enables the devices internal to its network to communicate with the external Internet via the same shared public IP address [1]. This function allows the internal configuration of a private network to be self-managed and requires only a single public IP address to be provisioned for the entire network. Furthermore, virtual private networks (VPNs) have become exceedingly popular to facilitate secured remote work [2], access geo-restricted content [3], and to conceal online malicious activity [4].

The topology of a typical private network is shown in Fig. 1. The devices internal to a private network access external services (e.g., E-mail and web browsing) through a mediating gateway (e.g., a network address translation (NAT) router) [1]. The mediating gateway obfuscates the internal characteristics of a private network through amalgamating the behaviour of its constituent devices. This obfuscation leads to decreased situational awareness of the devices within a private network.

Network situational awareness provides the necessary insight for a network operator or security analyst to investigate the behaviour of their network and its constituent devices [5]. The insight gained through situational awareness not only allows for the identification of vulnerabilities that may exist on the network but also allows for faster response times in the event of a cyber-attack [6]. It is therefore crucial to investigate techniques to facilitate a network operator's situational awareness within the private networks they administer.

It is often infeasible to implement a monitoring solution for each private network under analysis. Network operators may be restricted from deploying a monitoring solution within a private network due to resourcing constraints or due to the governance of the private network[1]. Furthermore, ad hoc private networks (e.g., mobile tethering) can be deployed to a network unbeknownst to a network operator and thus would remain un-monitored.

In recent years, device management solutions have been investigated to acquire situational awareness behind a private network through monitoring the communication link between the mediating gateway and the wider public Internet [7]–[10]. This point of analysis (Fig. 1) allows situational awareness behind private network to be acquired without requiring an internal monitoring solution.

Related work investigating situational awareness in private networks has assumed access to specific flow statistics (e.g., TCP timestamp and IP-ID [7]–[10]). These flow statistics are optionally provided when monitoring a network and thus their availability across diverse TCP/IP networks is not guaranteed. In contrast, we provide a technique to acquire insight into a private network utilising only the source and destination IP address fields. These two fields are necessary to describe a network flow [11]; and are therefore widely available across diverse network monitoring solutions.

To achieve the aim of this paper, we extend our previous work in representing network traffic as a bipartite graph [12], [13]. In particular, we leverage graph embeddings to transform the structural information contained in a bipartite graph into a d-dimensional latent space. Graph embeddings were chosen as they have been shown in other domains (e.g., natural language processing) to be semantically meaningful under linear combination [14]. In this paper, we reveal how this property can be exploited to identify the composition of device characteristics acting behind a private network.

We define a confidence metric to validate the analysis presented within this paper. This confidence metric addresses a severe limitation in related machine learning (ML) based solutions for private network analysis. That is, the related solutions do not provide insight into the uncertainty of their

This research is supported by the Australian Government Research Training Program (RTP) Scholarship and the Commonwealth of Australia as represented by the Defence Science & Technology Group of the Department of Defence.

[1]A private network may be governed by a separate organisation to the parent network.

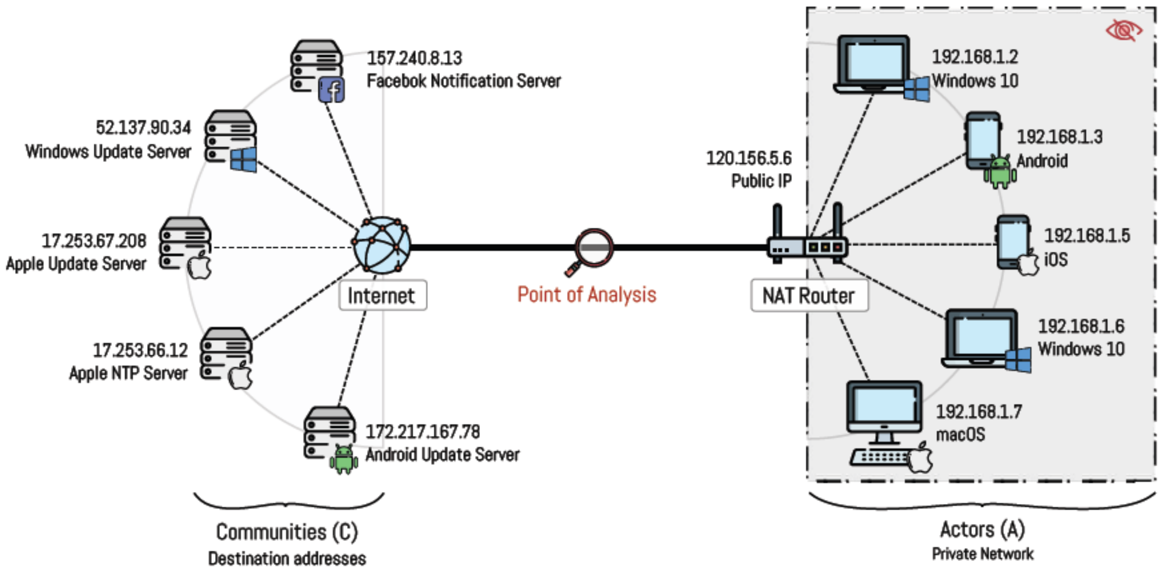


Fig. 1: The topology of a typical private network. The network address translation (NAT) router acts as a mediating gateway between the wider public Internet and the devices within the private network. The typical point of analysis becomes saturated as the size of the private network increases; thereby reducing a network operator's situational awareness within the private network.

predictions. In contrast, we provide a metric to inform the network operator as to whether the predictions made can be accepted or whether they require further analysis.

The key contribution of this paper is three-fold:

1) We validate the use of prediction intervals (PIs) and ensemble models to provide a confidence metric for ML-based network reconnaissance solutions. We show that this confidence metric accurately quantifies model and data uncertainty when estimating the composition of private networks. This confidence metric identifies when an ML-based solution should be retrained to adapt to changing network environments.
2) We provide PiPiN; a graph embedding technique that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. PiPiN exploits a novel insight into the vector decomposition of graph embeddings to characterise a private network based only on the source and destination IP address fields.
3) We show that PiPiN can accurately estimate the number of devices within a private network consisting of up to 500 devices. Furthermore, PiPiN is shown to estimate the composition of device manufacturers, operating systems, and applications that are used within a private network. PiPiN therefore provides a network operator with crucial insight into the characteristics of a private network.

The remainder of this paper is structured as follows: A brief survey of related work in characterising private networks is given in Section II. Section III introduces PiPiN; the proposed methodology for acquiring situational awareness within private networks. Section IV outlines the dataset used for our analysis. Section V provides an empirical analysis of the efficacy of PiPiN. The paper concludes in Section VI with a discussion of results and suggestions for future work.

II. Related Work

Several recent works have focused on characterising private networks and their constituent devices. The principal aim of these works are 1) to identify the presence of a private network—that is, identify whether an IP address is used by a single device or multiple—and 2) to characterise the devices within the private network.

A well-documented approach for characterising a private network is to examine the incremental identification fields in the traffic forwarded by the private network's NAT router (e.g., IP-ID and TCP Timestamp) [7]–[10]. Conflicting sequences within these fields can be used to indicate the presence of multiple devices. The principal benefit of this approach is that individual streams of traffic can be attributed to their corresponding device within the private network. This approach has limited applicability as it relies on the specific idiosyncrasies of each OS's TCP/IP protocol stack. For example, macOS randomly updates the IP-ID field and thus can not be identified by this field [7]. Additionally, NAT routers can be configured to modify incremental ID fields to counter-act the surveyed detection methods [7].

Payload identifiers have also been investigated to identify the presence of a private network [15]–[17]. These approaches check for conflicting information in device specific identifiers such as HTTP's user-agent field. If a conflict is found, it is assumed that there are multiple devices sharing the same IP address. Techniques based on payload identifiers show high classification performance as well as the ability to detect distinct OS versions acting within a private network.

Payload identifiers often underestimate the size of a private network as they cannot be used to distinguish between devices with similar characteristics [15]. This scenario is prevalent in enterprise networks where devices are commonly installed with identical configurations to ease their management. Additionally, the efficacy of payload-based approaches has drastically decreased in recent years due to the widespread adoption of encryption standards (e.g., within VPNs). It is therefore no longer feasible to rely on payload identifiers such as HTTP's user-agent field to characterise the devices operating within a private network.

Several recent works have investigated statistical features to infer the presence of a private network in the absence of payload identifiers [18]–[20]. The axiom of these approaches is that the activity of a private network is significantly higher than that of any single device acting individually. While the surveyed approaches were shown to detect the presence of a private network, they do not attempt to characterise the devices acting within the private network. The insight generated from the surveyed approaches is therefore limited.

We identify three critical limitations within the related work for generating situational awareness within a private network:

1) **Feature Availability**: There is no consensus as to which features are applicable for deployment across diverse TCP/IP networks [21]. For example, even when using the same standard—such as Cisco's NetFlow [22]—network operators heuristically select the features they use to monitor their networks. It is therefore likely that techniques which rely on a large number of heterogeneous features would be difficult to deploy across diverse TCP/IP networks.
2) **User Privacy**: The related techniques proved in [7]–[10] aim to de-anonymise the devices acting behind a private network. This aim is antithetical to a major application of private networks—to protect user privacy. The privacy of the users of the evaluated network should therefore be considered before the utilisation of such techniques.
3) **Confidence**: The related techniques do not provide a metric for the confidence of their prediction. An operator using such techniques would therefore have no indication if the technique they are using is functioning correctly. For example, changes to a network's behaviour (e.g., software updates) could invalidate the predictions made by a technique unbeknownst to the network operator.

These limitations would heavily restrict the deployment of the surveyed techniques for real-world application.

This paper aims to address the limitations found in the related work through a graph-theoretic approach. In our previous work [12], [13], [23], we have shown the feasibility of utilising graph structures to characterise the behaviour of a device through only monitoring the IP addresses it affiliates with. We exploit this widely available feature set to provide network reconnaissance solutions that are scalable, independent of encryption, and deployable across diverse TCP/IP networks.

In this paper, we provide the following extensions to our previous work to address the limitations of user privacy and confidence. First, we extended our previous technique to identify the size and composition of a private network. By focusing on the private network's composition, we mitigate the privacy concerns of previous techniques as the characteristics of the individual devices and users are not disclosed. Second, inspired by Pearce *et al.* [24], we provide a confidence metric for ML-based network reconnaissance solutions.

III. Method

The IP addresses that a device affiliates with provides significant insight into the characteristics of the device itself [12], [13], [23]. The significance of this insight has recently become widely applicable in TCP/IP networks due to a newfound reliance on cloud infrastructure to manage, notify, and update the software installed on a device [25]. For example, a device that communicates with a Windows update server is likely to be running a Windows OS [26]. It can be further postulated that multiple devices running the Windows OS will communicate with the Windows update server more frequently than any of the devices individually. By exploiting this axiom, we propose a technique to estimate the number and composition of devices operating behind a private network.

A weighted bipartite graph was used to represent the affiliations between devices and their affiliated Internet servers. This representation creates a graph with two distinct sets of vertices: 1) *actors* - the devices under analysis and 2) *communities* - the Internet services in which the devices communicate with (see Fig. 1). Edges in a bipartite graph exist only between vertices of the opposing set. In this representation, the weight of an edge represents the number of flows between a device and a server over the capture period.

Graph embeddings were utilised to generate insight from the bipartite graph structure. Embeddings were investigated as they have been shown in different domains to be semantically meaningful under vector combination. A well-known example of this property comes from natural language processing (NLP) whereby the vector combination of embeddings "King – Man + Woman" results in the embedding for the word "Queen" [14]. This example illustrates the significance of drawing insight from not only the embeddings themselves but also from their relevance to each other.

This paper provides the first known use of vector combination in the graph embeddings domain. We reveal that vector decomposition can be used to estimate the composition of devices acting behind a private network. A comparison of the aim of this work to the example first introduced in NLP is illustrated in Fig. 2.

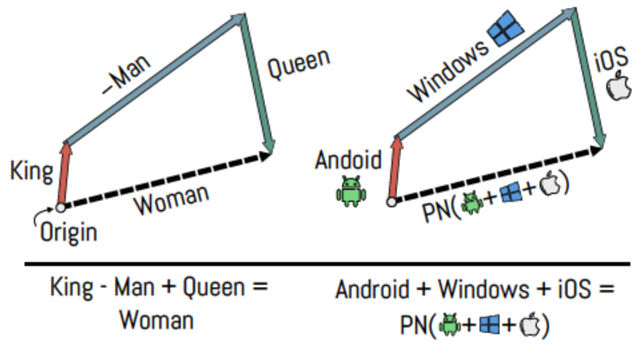

–Man
Queen
King
Woman
Origin
Windows
iOS
Andoid
PN(🤖+⊞+)
King - Man + Queen = Woman
Android + Windows + iOS = PN(🤖+⊞+)

Fig. 2: An illustration comparing the use of vector combination to analyse the embeddings generated by natural language processing (NLP) and the private network (PN) embeddings produced in this paper. We postulate that the embedding generated for a private network can be decomposed into a vector combination of the network's constituent devices.

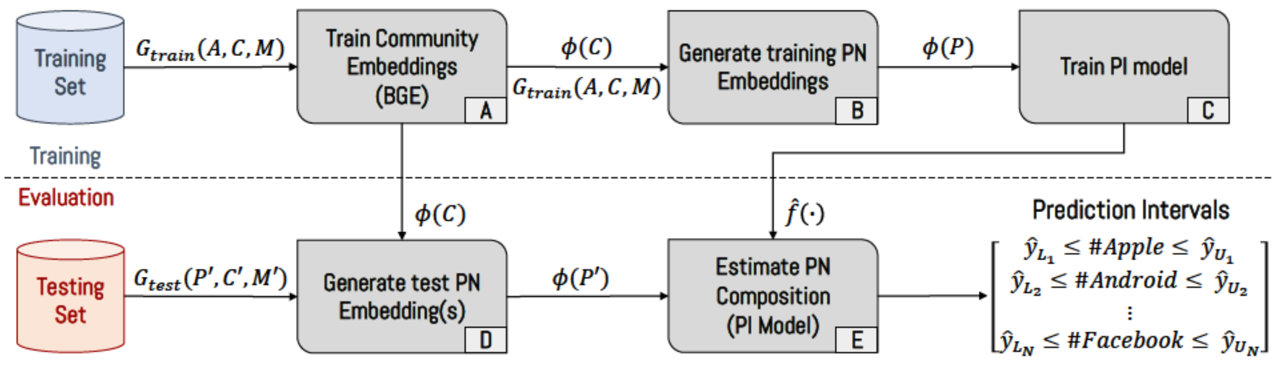

Training Set
$G_{train}(A,C,M)$
Train Community Embeddings (BGE)
A
$\phi(C)$
$G_{train}(A,C,M)$
Generate training PN Embeddings
B
$\phi(P)$
Train PI model
C
Training
Evaluation
$\phi(C)$
$\hat{f}(\cdot)$
Prediction Intervals
Testing Set
$G_{test}(P',C',M')$
Generate test PN Embedding(s)
D
$\phi(P')$
Estimate PN Composition (PI Model)
E
$\hat{y}_{L_1} \leq \#Apple \leq \hat{y}_{U_1}$
$\hat{y}_{L_2} \leq \#Android \leq \hat{y}_{U_2}$
$\hat{y}_{L_N} \leq \#Facebook \leq \hat{y}_{U_N}$

Fig. 3: PiPiN: The proposed methodology to estimate the composition of devices acting behind a private network. PiPiN utilises graph-based machine learning and Prediction intervals to confidently acquire situational awareness behind a Private Network.

Fig. 3 illustrates PiPiN; the proposed methodology to investigate a private network through the vector decomposition of its embedding. PiPiN is divided into two principal stages—training and evaluation. In the training stage, the embeddings for the community set are first generated (Section 3-A). These community embeddings are then used to create a training set of example private network (PN) embeddings (Section 3-B). Finally, the prediction interval (PI) Model [24] is trained to estimate the characteristic composition of a private network from its embedding (Section 3-C). In the evaluation stage, the embeddings for the private network(s) in the testing set are created (Section 3-D). The evaluated PN embedding(s) are then decomposed using the trained PI Model model (Section 3-E).

A. Train community embeddings

The community set is embedded to 1) reduce the dimensionality of the resultant private network embedding and 2) encode the similarity between communities. In this work, two communities are similar if they exhibit a proportionality high overlap in their neighbourhood sets. This definition states that Internet services (*i.e., communities*) are similar if they are frequented by a similar set of devices (*i.e., actors*). The intuition behind this definition is derived from the observation that Internet services acting in similar roles (e.g., Windows update servers) will be frequented by a similar set of devices.

The community embeddings are trained on a bipartite graph, $G_{train}(A, C, M)$; where, A is a set of individual devices, C is the Internet services contacted during the capture period, and M is the set of affiliations between the devices and Internet services. The affiliations between Internet services and individual devices must be used in the training set to encode the similarity of Internet services within the community embeddings.

The community embeddings are trained using bipartite graph embeddings (BGE) [23]. BGE utilises a fully-connected neural network to embed the structure of a bipartite graph into a set of actor and community embeddings. The architecture of BGE is illustrated in Fig. 4. In this work, only the community embeddings are retained as they are required to generate the private network (PN) embeddings.

B. Generate private network embeddings (training)

A diverse set of labelled PN embeddings are required to train the PI model (Section III-C). To generate these compositions, devices are sampled from the training set and their respective network behaviour is amalgamated to form a private network. PiPiN is able to provide labelled examples of vastly distinct network compositions by sampling devices from the training set. This sampling procedure would be difficult to replicate for techniques that rely on the specific idiosyncrasies of a private network. For example, techniques that investigate IP-ID sequences would have to ensure that their amalgamation accurately reflects how a real private network processes such sequences. In contrast, our technique only requires the modification of a device's local IP address.

A private network's embedding must encode 1) the characteristics of the private network and 2) its level of activity. The characteristics of a private network determines the types of devices or applications that may exist within the private network; whereas, its activity level determines the size of the private network (e.g., the number of devices or applications).

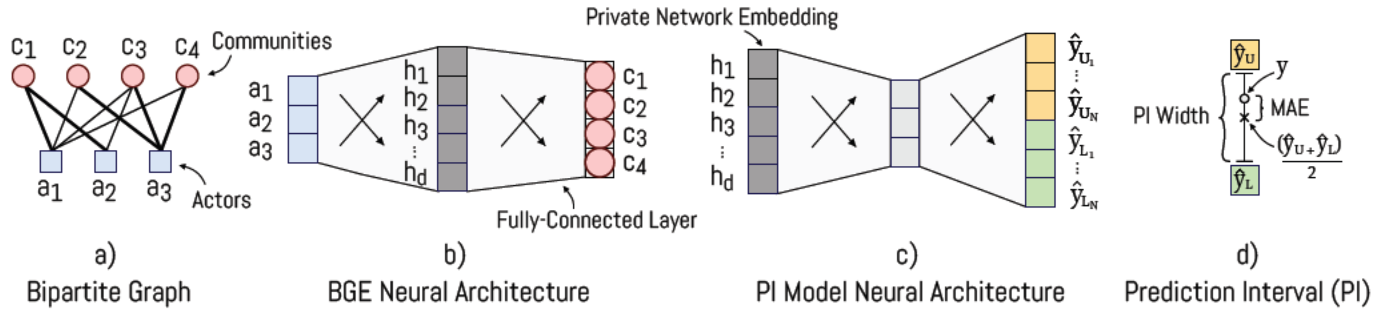
The characteristics of a private network is determined by the communities in which it frequents. For example, a private network that only communicates with Windows update servers is likely to be composed of devices only running the Windows OS. These characteristics are encoded in the community embeddings generated in Section III-A.

In this paper, the activity level of a private network is determined by the number of flows between a private network and the community set. Term frequency-inverse document frequency (TF-IDF) [27] is used to ensure that commonly frequented communities do not dominate the information contained within a private network's embedding.

The embeddings for a private network are generated through the linear combination of the private network's weighted memberships within the community set as follows

$$\phi(P) = \phi(C)M \tag{1}$$

where $\phi(P)$ is the set of the training PN embeddings. $\phi(C)$ are the community embeddings, and M is the incidence matrix of a bipartite graph that links the private networks to the



Communities
Actors
Private Network Embedding
Fully-Connected Layer
PI Width
MAE
a) Bipartite Graph
b) BGE Neural Architecture
c) PI Model Neural Architecture
d) Prediction Interval (PI)

Fig. 4: The model architecture of bipartite graph embeddings (BGE) [23] and the prediction interval (PI) model. BGE is used to encode the embeddings for the community set of a bipartite graph. The community embeddings are then used to define the embeddings for a set of private networks (PNs). The PI model is then used to predict the composition of a private network by estimating the upper and lower bounds of each characteristic of interest (e.g., the size of a private network). Mean absolute error (MAE) is used to determine the distance of the PI to the true value, y.

community set. The weights of the incidence matrix, M, are determined by the number of flows between the private networks and the community set scaled by TF-IDF.

C. Train PI model

Given a private network's embedding, we aim to predict the number of devices within the private network that exhibit each of the characteristics of interest. This task can be treated as a multi-output regression problem as follows

$$y = f(x) \tag{2}$$

where, y is a vector denoting the true number of devices per characteristic of interest, x is the embedding of the private network, and $f(\cdot)$ is an ideal mapping function.

There are two limitations with this idealised regression problem. First, there may not be a single mapping between y and x to allow for an optimal mapping function to be created. Second, there is no indication of the fidelity of the mapping function when evaluated on new data. To solve these two limitations, we extend the ideal regression problem by considering data uncertainty and model uncertainty, respectively.

1) Data Uncertainty: It is expected that a significant variation would exist between the embeddings of a set of private networks even if they contain similar characteristic compositions. This variation is derived from the individual behaviour of the devices operating on each private network. For example, two private networks with the same underlying characteristics could exhibit vastly distinct network behaviour simply due to the user activity on each private network. This variation would produce two distinct private network embeddings, x_1 and x_2, that would both map to the same target output, y.

Fig. 5 provides an example of the variation of the generated PN embeddings. The figure compares the size of a selected private network to that of the five nearest private networks within the embedding space. It is seen that for a given PN embedding, there is a noticeable variation in the private network's size in relation to its nearest neighbours. This example illustrates the effect of data uncertainty within eq. (2).

To account for data uncertainty, we introduce a variance term within the eq. (2) as follows

$$y = f(x) + \sigma_{data} \tag{3}$$

where, σ_{data} is the uncertainty of a prediction due to the variance in the private network embeddings.

To account for σ_{data}, we evaluate the use of prediction intervals (PIs) for the analysis of a private network's embedding. A PI provides an estimate of the upper and lower bounds of the true output for a pre-specified probability. The aim of a PI can thus be defined as estimating an upper and lower bound that satisfies the following

$$\Pr(\hat{y}_L \leq y \leq \hat{y}_U) \geq P_c \tag{4}$$

where probability is defined by $\Pr(\cdot)$; P_c is a pre-specified probability; and $\hat{y}_L$ and $\hat{y}_U$ are the estimated lower and upper bounds of the prediction interval, respectively.

The use of a PI rather than an absolute estimate allows for variance to exist within the private network embeddings without negatively affecting the optimisation of the regression model. Furthermore, the use of absolute estimates can often be misleading when deployed in real-world applications as they provide a false sense of confidence [24]. In contrast, a PI

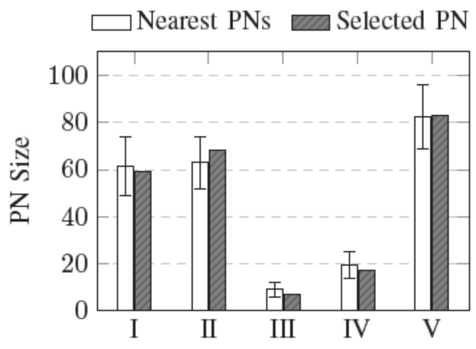


Fig. 5: An example of the variation in the private network (PN) embeddings. Five private networks of various sizes were sampled from the t_0 capture (Section IV). For each private network, the five nearest private networks within the embedding space were sampled.

enables a network operator to make more informed decisions by identifying the range of possible device compositions that may exist within a private network.

A PI regression model designed by Pearce *et al.* [24] was used to acquire insight into a private network through analysis of its embedding. In particular, a fully-connected neural network was trained with the loss function as specified in Algorithm 1. The original loss function as specified by Pearce *et al.* [24] remains largely unaltered; however, an extension was made to allow for multi-output regression tasks. This extension defined the loss function as the sum of the individual loss functions (as defined by Algorithm 1) computed for each characteristic of interest.

An illustration of the multi-output PI model's neural architecture is provided in Fig. 4. It is seen that for each characteristic of interest, two output nodes are required to denote the upper and lower bounds of the PI. The upper and lower and bounds are trained using the loss function stated in Algorithm 1 and stochastic gradient decent (SGD) [28].

2) Model Uncertainty: It is impractical to create a training set that would cover all unique combinations of devices, software, and user behaviour that could exist behind a private network. Furthermore, network conditions may change after a model has been trained which may invalidate the model unbeknownst to the operator. It is therefore essential to account for the uncertainty of the model when evaluating unknown inputs.

An ensemble model was utilised to quantify model uncertainty as suggested within [24]. Ensemble models quantify model uncertainty through measuring the variation in a set of trained models' predictions. Small variations indicate that the evaluated data is well represented within the training set and hence a greater level of confidence is exhibited. Large variations are an indication that the models have not learned an appropriate mapping function for the evaluated data and hence there exists uncertainty in their prediction.

We quantify model uncertainty by measuring the variance in the upper and lower bounds of m models contained within the ensemble model. Where, the upper and lower bounds of the ensemble model are derived as follows

$$\overline{y}_{Ui} = \frac{1}{m}\sum_{j=1}^{m} \hat{y}_{Uij}, \tag{5}$$

$$\sigma_{Ui}^2 = \frac{1}{m-1}\sum_{j=1}^{m} (\hat{y}_{U}ij - \overline{y}_{Ui})^2, \tag{6}$$

$$\tilde{y}_{Ui} = \overline{y}_{Ui} + 1.96\sigma_{Ui} \tag{7}$$

where $\hat{y}_{Uij}$ is the upper bound produced by model j for a prediction i. The ensemble's lower bound, $\tilde{y}_{Li}$ is produced similarly, however, the corresponding variance is subtracted rather than added as follows

$$\tilde{y}_{Li} = \overline{y}_{Li} - 1.96\sigma_{Li} \tag{8}$$

The model uncertainty for a given prediction, i, is then quantified by the average variance of the ensemble's upper and lower bounds as follows

$$\sigma_i = \frac{\sigma_{Ui} + \sigma_{Li}}{2} \tag{9}$$

The variance is calculated for each PI. The model uncertainty for each characteristic of interest is therefore defined for each PN embedding.

D. Generate private network embeddings (testing)

The testing set is composed of the private network(s) observed from a network capture. PN embeddings are then generated with the same procedure as detailed in Section III-B.

The observed private networks are represented by a bipartite graph constructed from their network activity, $G_{test}(P', C', M')$; where, P' is the set of observed private networks, C' is the communities contacted during the capture period, and M' is the set of weighted memberships (i.e., number of flows) between the set of private networks and the communities they affiliate with over the Internet.

Algorithm 1 Prediction Interval (PI) Loss Function [24]

```
Input true vector, $y$; predicted upper and lower bounds, $\hat{y}_U$ and $\hat{y}_L$, respectively; coverage factor[2], $\alpha$; sigmoid softening factor[3], $s$; and a biasing factor[4], $\lambda$.

# Hard cut-off; where, sgn denotes the signum function.
$k_{HU} = \max(0, \text{sgn}(\hat{y}_U - y))$
$k_{HL} = \max(0, \text{sgn}(y - \hat{y}_L))$
$k_H = k_{HU} \odot k_{HL}$

# Soft cut-off.
$k_{SU} = \text{sigmoid}((\hat{y}_U - y) \cdot s)$
$k_{SL} = \text{sigmoid}((y - \hat{y}_L) \cdot s)$
$k_S = k_{SU} \odot k_{SL}$

# The mean prediction interval width (MPIW).
$MPIW =$
    $\text{reduce\_sum}((\hat{y}_U - \hat{y}_L) \odot k_H)/\text{reduce\_sum}(k_H))$

# The prediction interval coverage probability (PCIP).
$PICP = \text{reduce\_mean}(k_S)$

# The utilised loss function; where, $n$ denotes the selected batch size[5] used for training.
$Loss = MPIW + \lambda \cdot \frac{n}{\alpha(1-\alpha)} \cdot \max(0, (1-\alpha) - PICP)^2$
```

[1]The coverage factor, α, determines the percentage of expected samples that should be contained with the PI. For example, a typical coverage factor of $\alpha = 0.05$ states that 95% of expected samples should be contained in an optimally defined PI.

[2]The sigmoid softening factor, s, determines the sharpness in the boundary for which a sample is considered to be internal or external to a PI. $s = 160$ was used within this work as suggested in [24].

[3]The biasing factor, λ, controls the importance of maintaining training samples within the PI in contrast to reducing the PI's width. $\lambda = 5$ was empirically found to produce desirable results within this evaluation.

[4]It was found that setting $n = 1$ in the loss function was useful when training with larger batch sizes (i.e., > 32); otherwise, n grows to dominate the loss function. In particular, when $n \gg \lambda$.

E. Estimate PN composition

The culminating step of the evaluation stage is to decompose the embedding generated for the private network(s), $\phi(P')$, using the trained PI model, $\hat{f}(\cdot)$. This stage estimates the upper and lower bounds for the number of devices likely to exhibit each characteristic of interest. Furthermore, the model's confidence for each upper and lower bound is provided. This insight provides a network operator with the necessary insight to make informed decisions regrading the security and management of the private networks that they administer.

IV. Dataset

To evaluate PiPiN, three network captures were taken on a university's wireless network. This network allows both staff and students alike to connect their wireless devices into the university's enterprise network. It therefore provides our analysis with insight into the realistic behaviour of user devices.

Three 26-hour captures were taken on the university's wireless network. These captures were taken three months apart on the 1st of May (t_0), August (t_1), and November (t_2) of 2019. For all analysis presented within this paper, the t_0 capture was used as the training set; whilst all three captures captures were used as the testing sets. This evaluation was conducted to investigate the longitudinal efficacy of PiPiN.

The distribution of manufacturers, operating systems (OSs), and applications used within the three captures is shown in Table I. This evaluation considered six applications: Facebook, WeChat, Twitter, Gmail, Spotify, and Netflix. These applications where chosen as 1) they are well-known and popular applications, 2) represent distinct application types ranging from social media to video streaming, and 3) they were well represented within the evaluated network.

To simulate the behaviour of a private network, the network traffic of a set of devices was amalgamated such that they appear as originating from a singular device. Amalgamating the network traffic in this manner has been shown to provide an accurate approximation of a private network [7], [8], [16].

V. Results

PiPiN was evaluated on several scenarios relevant for the analysis of private networks. In particular, the size of a private network is estimated in Section V-A. Section V-B estimates the device composition of a private network (i.e., the device manufacturer and OS composition). The composition of applications used within a private network is then estimated in Section V-C. The results section is concluded in Section V-D with a discussion on how to utilise the proposed confidence metric to identify uncertainty in the predictions made by PiPiN.

A. Estimating Private Network Size

This evaluation investigates the use of the PiPiN to predict the number of devices acting within a private network. A device is defined as a system that either generates—or is the recipient of—an observed network communication. Within this evaluation, each device represents a physical device operating within the captured network; however, this representation could also extend to a virtual machine (VM) running within a physical device.

The knowledge of how many devices are operating within a private network is beneficial for several reasons. First, it advises the provision of resources to manage the private network appropriately [9]. This benefit is essential within large enterprise networks and Internet service providers (ISPs) which often manage numerous individual private networks. Second, it assists in enforcing appropriate cybersecurity practices through an increased awareness of the activity within the private network. For instance, a significant change in the number of devices within a private network may be an indication of anomalous behaviour in certain network environments.

Four datasets were created to evaluate the use of PiPiN to estimate the number of devices within a private network. First, a training set was created from generating various sized private networks sampled from the devices within the t_0 capture. The number of devices in each private network was randomly selected between 1 and 500 to evaluate PiPiN over a large range of private network sizes. An embedding was then generated for each private network as discussed in Section III-B. In total, the training set was composed of 2^{14} embeddings of various sized private networks. Three testing sets were then created on the t_0[6], t_1, and t_2 captures using the same procedure.

The performance of PiPiN on the testing sets is shown in Fig. 6. Within subfigures a)-c), PiPiN is shown to provide an accurate PI for the various sized private networks within each testing set. In these subfigures, PiPiN is shown to exhibit a significant degree of uncertainty in private network size as indicated by the large PI widths. This result is expected due

TABLE I: The distribution of devices on the university's wireless network. Network captures were taken on the 1st of May (t_0), August (t_1), and November (t_2) of 2019.

Characteristics		t_0 01/05/2019	t_1 01/08/2019	t_2 01/11/2019
Manufacturers	Apple	320	292	305
	Intel	33	32	27
	Samsung	54	42	47
	Huawei	19	15	10
	OPPO	10	10	8
OS	iOS	237	216	232
	Android	118	105	102
	Windows	43	39	32
	macOS	83	75	73
Applications	Facebook	313	241	271
	WeChat	49	58	51
	Twitter	68	33	53
	Gmail	192	156	153
	Spotify	39	43	53
	Netflix	16	14	13

[6]The training and testing sets created from the t_0 capture are composed of distinct private networks.

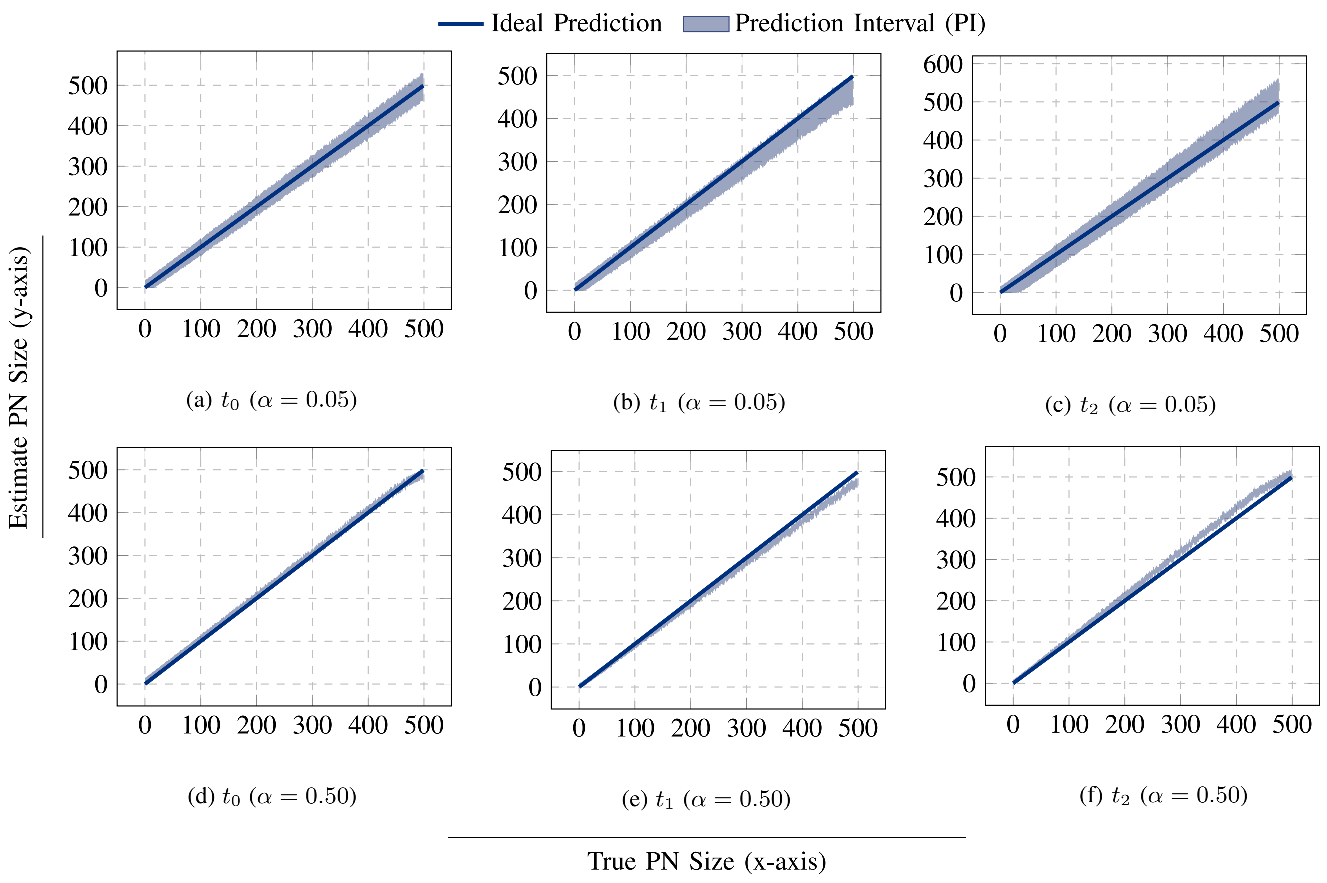


Fig. 6: Predicted private network size on the testing sets. Subfigures a) - c) provide a prediction interval with a 95% confidence ($\alpha = 0.05$). Subfigures d) - f) provide a prediction interval with a 50% confidence ($\alpha = 0.50$).

to the large diversity of user operated devices that are active within the evaluated network. A single device may be unused during the capture period or involved in network intensive tasks such as video streaming. The PI bounds were shown to broaden to account for this variance at a 95% confidence level (α=0.05).

A trade-off between the PICP and the MPIW can be tuned through two hyperparameters; the biasing factor, λ, and the coverage factor, α. Tuning these two hyperparameters may be necessary to mitigate the broadening of the MPIW to accommodate outliers in an evaluated data set. Subfigures d)-f) show the result of PiPiN evaluated at 50% confidence level ($\alpha = 0.50$). The result of this evaluation shows that PiPiN can provide an accurate and precise estimate of the number of devices within private networks up to a size of approximately 200 devices; after which, PiPiN's accuracy starts to decline. PiPiN's PI, however, never completely deviates from the true number of devices. This result indicates that the use of PiPiN at a 50% confidence level would still provide a useful estimate of the number of devices for even significantly large private networks.

The highest performance was seen on the t_0 testing set. This result supports the expectation that a more accurate PI would be generated for a private network composed of devices that are represented within the training set. PiPiN, however, is shown to still provide accurate PIs on the t_1 and t_2 testing sets. This result illustrates that PiPiN can provide an accurate prediction of the number of devices within a private network consisting of unknown devices. Furthermore, the performance of PiPiN on the t_1 and t_2 testing sets show that the technique can remain effective for at least six months after its initial training.

The result of this evaluation shows two potential use-cases for the deployment of PiPiN. First, a confident approximation of the number of devices could be estimated using PiPiN at 95% confidence level. This estimation could be used to cluster similarly sized private networks and enforce appropriate managerial policies. Second, PiPiN's confidence level could be reduced to provide a more precise estimate of the number of devices within a private network. It was shown that PiPiN evaluated at a lower confidence can produce both accurate and precise estimates of the size of a private network consisting of 200 devices or fewer[7]. This precise estimation would allow a security analyst to detect smaller changes within a private network that may be indicative of anomalous use.

B. Manufacturer and Operating System Composition

This evaluation investigates the use of PiPiN to estimate the composition of device manufacturers and operating systems

[7]Private networks of 200 devices or fewer would account for most typical private networks such as those used by households and small offices.

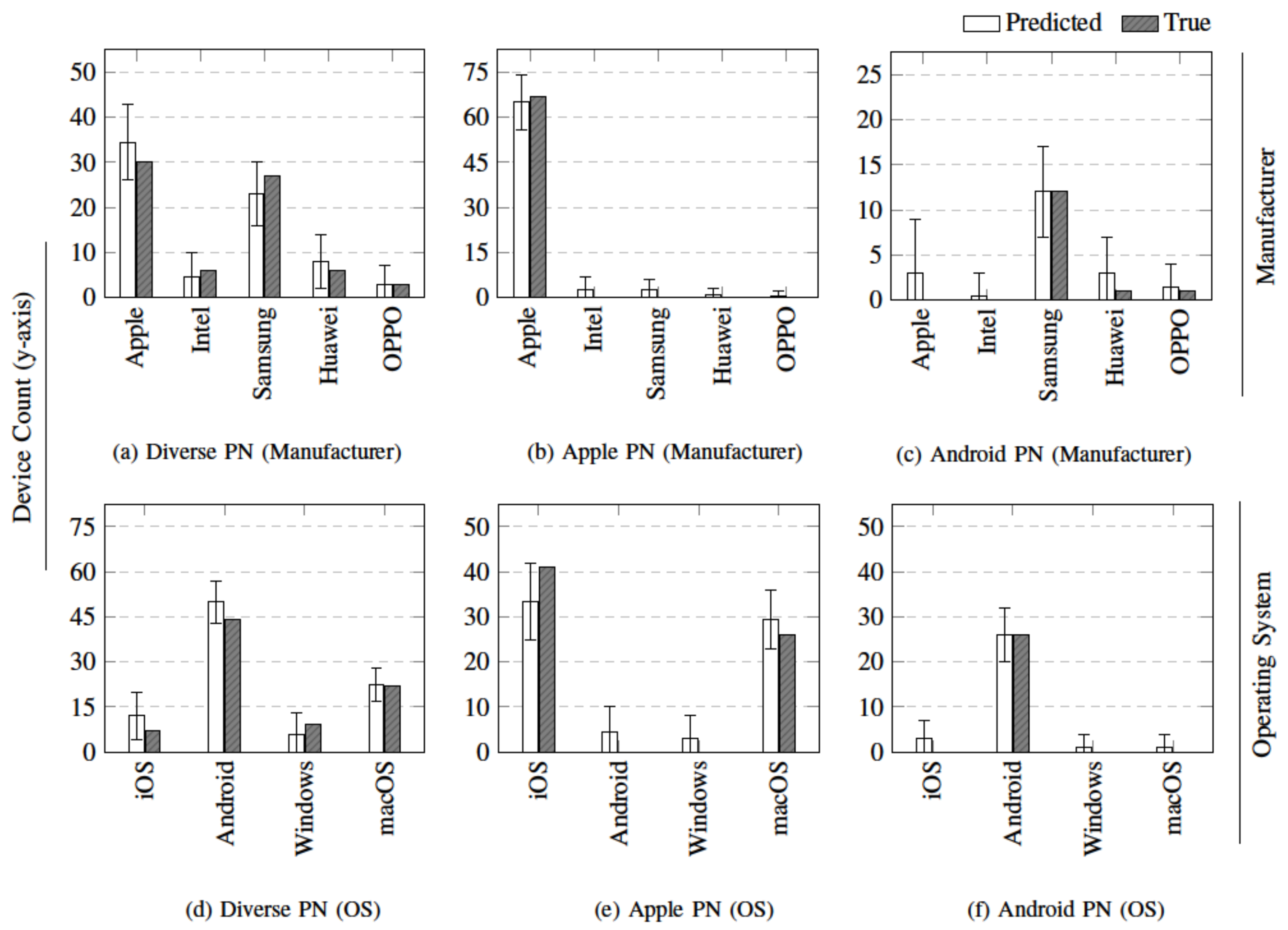


Fig. 7: Predicted manufacturer and operating system (OS) composition of distinct private networks. All predictions shown were evaluated on the t_2 testing set. Subfigures a) and d) evaluate a private network of diverse manufacturers and OSs; subfigres b) and e) evaluate a private network composed only of Apple manufactured devices; and subfigures c) and f) evaluate a private composed only of devices running the Android OS.

(OSs) behind a private network. A device's manufacturer is defined as the company responsible for its physical construction; whereas, a device's OS is the primary software used to manage and interact with the device.

The knowledge of a device's manufacturer allows for an estimation of its security and managerial requirements. For example, a device manufactured by Apple has a distinct set of vulnerabilities as opposed to a device manufactured by Huawei. Furthermore, the utilisation of a private network can often be identified through the uniformity of its manufacturer composition. For example, a computer lab is likely to consist of a singular device manufacturer (e.g., Intel[8]).

A device's OS provides a similar insight as to its manufacturer. OSs, however, provide additional information useful to identify more specific security and managerial policies. For example, certain vulnerabilities may exist on devices running macOS that are not present for those running iOS even though they are both manufactured by Apple. Alternatively, the same policies may need to be applied for all devices running a particular OS (e.g., Android) regardless of their manufacturer (e.g., Huawei, Samsung, or OPPO).

The same process as described in Section V-A was used to generate the training and testing sets to evaluate PiPiN for estimating manufacturer and OS composition. Devices were sampled, however, by their respective OS to mitigate the impact of class imbalance. The resultant private networks contained between 0-50 devices of each evaluated OS for a total possible size of 200 devices.

The overall performance of PiPiN for manufacturer and OS composition on the testing sets are shown in Table II and Table III, respectively. The results on t_0 testing set show that PiPiN can accurately identify the composition of manufacturers and OSs acting behind a private network. Furthermore, the results on t_1 and t_2 testing tests show that PiPiN remains effective for the analysis of private networks for up to six months after training, however, there is a noticeable reduction in accuracy. This reduction is expected due the changing network behaviour of the devices under analysis (e.g., software updates).

The true and estimated manufacturer and OS compositions of three private networks are shown in Fig. 7. This figure illustrates the performance of PiPiN on individual private networks. The private networks were selected from the t_2

[8]Devices can be composed of hardware from various manufacturers. We have denoted Intel as the manufacturer of Intel-powered devices.

testing set and show the performance of PiPiN on distinct manufacturer and OS compositions. Within Fig. 7, a) and d) are an example of a private network used by a diverse set of manufacturers and OSs; b) and e) are an example of a private network used only by Apple manufactured devices; and c) and f) are an example of a private network used only by devices running the Android OS.

From Fig. 7, PiPiN is seen to enable the estimation of manufacturer and OS composition of distinct private networks. This result would allow a network operator to apply tailored security and managerial policies to administrate the devices operating within the private network. Furthermore, the ability of PiPiN to identify private networks composed of a singular manufacturer or OS is critical to identify how the network is being utilised.

C. Application Composition

This evaluation investigates the use of PiPiN to estimate the composition of applications used by the devices within a private network. In this work, applications are defined as the user-installed software running on a device. In particular, six applications were investigated: Facebook, WeChat, Twitter, Gmail, Spotify, and Netflix.

The knowledge of which applications are running on a device provides useful insight to a network operator. First, it identifies the utility of the device. For example, a device running distinct network applications (e.g., social media, E-mail, and video streaming) is more likely to be a user device rather than a printer or server [29]. Second, it is necessary to identify the risk of a device to the network as each application introduces its own unique vulnerabilities [30]. Third, it allows for the deployment of policies to better manage the overall network [31]. For example, an ISP may prioritise the network traffic associated with certain applications (such as video conferencing) to improve the quality of service (QoS) on the network.

The same process as described in Section V-B was used to generate the training and testing sets to evaluate PiPiN for estimating a private network's application composition. Devices were sampled, however, by their respective applications to reduce the impact of application class imbalance. The resultant private networks contained between 0-50 examples of devices running each of the evaluated applications[9].

Table IV depicts the performance of PiPiN to estimate the application composition of a private network. The high accuracy in the t_0 testing set indicates that the applications used by a private network can be inferred through the use of PiPiN. PiPiN, however, experienced a significant degree of uncertainty in application decomposition as indicated by the width of its PIs.

An example of the increased PI width is illustrated in Fig. 8. In Fig. 8 a), a confident prediction of the application composition is shown with relatively small PI widths; In Fig. 8 b), however, the size of WeChat's PI width would reduce the insight gained from this application's prediction.

[9]The number of each application within a private network could exceed 50 as each sampled device is likely to have multiple applications installed.

The reason for the increased PI width in Fig. 8 b) is identified through plotting its model uncertainty. The model uncertainty when estimating WeChat was significantly higher than any other application within this private network. This uncertainty causes the PI width to expand as the biasing factor, λ, is set to favour accuracy over smaller PI widths (Algorithm 1). Decreasing PiPiN's model uncertainty is therefore likely to improve its efficacy for application decomposition.

Model uncertainty infers a lack of representation within the training set. The training set evaluated in this section was created by sampling device behaviour to generate private networks of differing application compositions. The limitation of this method is that it does not sample the behaviour of each application individually. This may cause underrepresented application compositions when evaluated on a testing set.

To mitigate model uncertainty, a larger dataset should be generated such that PiPiN is trained on a greater diversity of private network compositions. Additionally, the behaviour of individual applications could be identified through filtering the network traffic based on the application rather than the device. This would allow for more diverse compositions of applications to be represented within the training set.

Table IV shows a significant degradation in PiPiN's accuracy with respect to time. This result is derived from the volatility in network behaviour exhibited by the evaluated applications. In comparison to manufacturers and OSs, applications tend to be updated more frequently and utilise more dynamic content delivery networks (CDNs). To mitigate this effect, the technique would have to be re-trained more frequently to keep pace with the dynamic behaviour of application activity. Model uncertainty would therefore provide a mechanism to indicate when PiPiN requires retraining as shown in the next section.

D. Model Uncertainty

In Section V-C, the effect of model uncertainty for application decomposition was identified. In this section, we further evaluate the correlation between model uncertainty and the accuracy of PiPiN. We show how model uncertainty could be used to inform a network operator as to when to reject the predictions made by PiPiN.

To investigate the response of model uncertainty, the evaluation provided in Section V-A was repeated with two alterations. First, the maximum size of the private networks in the testing sets were restricted to 100 devices. This restriction aimed to mitigate the effect of data uncertainty experienced at larger PN sizes. Second, two training sets was created. The first training set restricted the maximum size of the sampled private networks to 50 devices and the second to a maximum size of 100 devices.

Model uncertainty will increase when the model is evaluated on samples that were not well represented within the training set. A model trained on the first training set should therefore exhibit a noticeable elevation in its uncertainty when applied to private networks of more than 50 devices.

Fig. 9 illustrates the performance of PiPiN for estimating a private network's size on the t_1 testing set. The training of

TABLE II: Overall performance of PiPiN to estimate the manufacturer composition on the t_0, t_1, and t_2 testing sets. Accuracy reports the percentage of true values bounded by their respective PI, PI Width reports the average width of the PIs, and the mean absolute error (MAE) reports the average distance between the center of a PI and the true value.

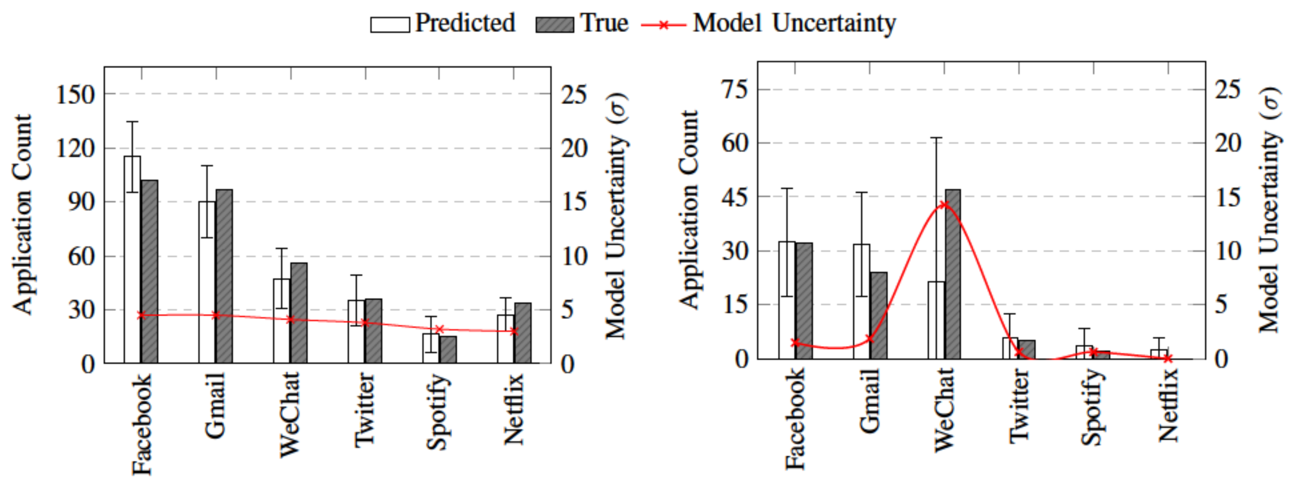
Manufacturers	t_0			t_1 ($t_0 + 3$ Months)			t_2 ($t_0 + 6$ Months)		
	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE
Apple	0.99	21.9	4.9	0.89	20.7	7.1	0.94	22.1	6.5
Intel	0.99	12.5	2.4	0.95	14.1	3.4	0.81	12.8	4.9
Samsung	0.98	9.7	1.8	0.96	10.8	2.7	0.97	12.6	3.2
Huawei	0.99	5.5	1.1	0.99	5.9	1.3	0.94	6.6	1.9
OPPO	0.98	3.4	0.7	0.98	3.7	0.9	0.99	4.3	0.9

TABLE III: Overall performance of PiPiN to estimate the operating system (OS) composition on the t_0, t_1, and t_2 testing sets.

Operating Systems	t_0			t_1 ($t_0 + 3$ Months)			t_2 ($t_0 + 6$ Months)		
	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE
iOS	1.00	20.2	4.7	0.82	17.4	7.4	0.86	17.4	6.6
Android	0.99	14.1	2.8	0.90	15.6	4.8	0.82	17.2	6.5
Windows	0.98	14.2	2.8	0.97	16.5	3.7	0.85	16.0	5.5
macOS	0.99	13.4	2.7	0.88	16.5	5.4	0.87	17.8	5.9

TABLE IV: Overall performance of PiPiN to estimate the application composition on the t_0, t_1, and t_2 testing sets.

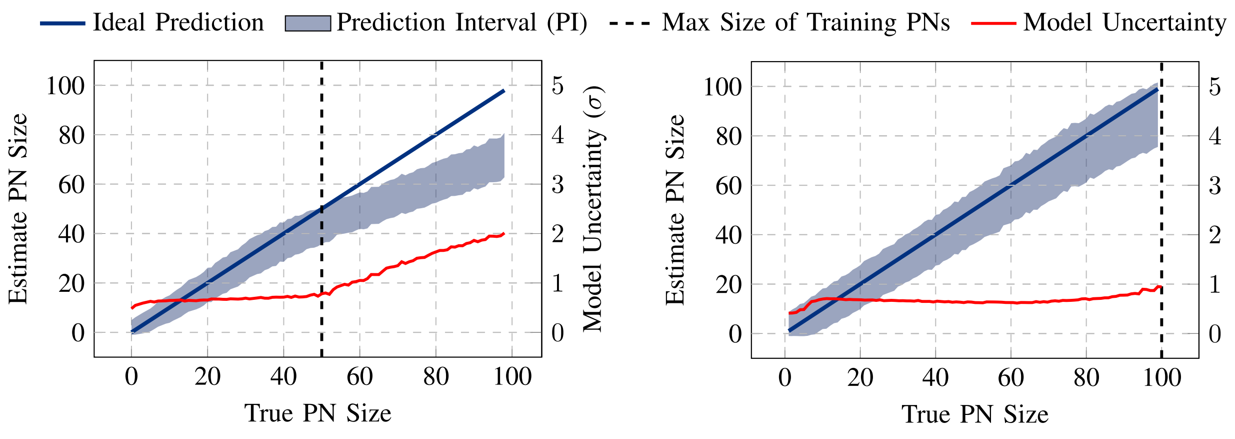
Applications	t_0			t_1 ($t_0 + 3$ Months)			t_2 ($t_0 + 6$ Months)		
	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE	Accuracy	PI Width	MAE
Facebook	0.99	30.8	5.0	0.85	36.1	10.7	0.71	34.5	14.4
Gmail	0.99	29.0	5.0	0.84	37.6	12.1	0.70	35.5	14.9
Wechat	1.00	22.0	4.0	0.93	30.7	9.4	1.00	46.0	4.4
Twitter	0.98	21.4	4.2	0.98	46.9	14.7	0.97	27.2	6.0
Spotify	0.99	17.5	3.3	0.79	22.2	8.7	0.60	18.3	9.9
Netflix	0.99	13.9	2.7	0.54	15.6	13.3	0.76	15.1	6.3



(a) Confident Application Prediction

(b) Application Prediction with Uncertainty

Fig. 8: Performance of application classification on two private networks within the t_1 dataset. a) provides an example of a desirable decomposition of the applications on a private network. b) provides an example where the composition of a specific application—in this case, WeChat—cannot be precisely determined. The uncertainty of the model on the WeChat application causes the PI width to grow significantly large.



(a) Training PNs limited to 50 devices

(b) Training PNs limited to 100 devices

Fig. 9: The effect of model uncertainty to estimate the size of a private network on the t_1 testing set. The training of PiPiN was restricted to private networks of size 50 or below in a) and 100 or below in b). Model uncertainty is seen to increase as the number of devices exceeds the maximum size contained within the training set. Furthermore, an increase in model uncertainty is shown to provide an indication of a decrease in the accuracy of the model. That is, the true output falls outside of the PI's upper and lower bounds.

PiPiN was restricted to private networks of size 50 or below in Fig. 9 a) and 100 or below in Fig. 9 b). The contrast between the model uncertainty in the two evaluations is evident. A sharp increase in model uncertainty is shown when applied to private networks that were not represented within the training set. Furthermore, an increase in uncertainty is shown to be correlated with a decrease in the accuracy of the model. That is, the true output falls outside of the PI's upper and lower bounds.

The result presented in Fig. 9 highlights the utility of model uncertainty for real-world deployment. Without model uncertainty, an operator would have no way of telling which model—Fig. 9 a) or b)—should be trusted. Furthermore, an increase in a model's uncertainty could be used to indicate to the operator when the model requires retraining. Model uncertainty is therefore a necessity for maintaining effective models within extended real-world deployments.

VI. Conclusion and Future Work

We provide PiPiN; a novel graph-based machine learning technique that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. This technique relies only on the availability of the source and destination IP addresses. PiPiN is therefore encryption invariant and can be widely deployed across diverse TCP/IP networks.

We show that PiPiN can accurately estimate the number of devices within a private network consisting of up to 500 devices. Furthermore, we show that PiPiN can estimate the composition of device manufacturers, operating systems, and applications that are used within a private network. PiPiN therefore provides a network operator with crucial insight into the characteristics of a private network.

We quantify a confidence metric to further the real-world applicability of PiPiN. Through this confidence metric, we ensure that changes in the observed network behaviour are made known to the network operator. This insight would be essential for maintaining effective models within extended real-world deployments.

In future work, we aim to address the applicability of PiPiN to identify the application composition of a private network. In Section V-C, we identify that a more comprehensive training set may help to reduce the model uncertainty for application decomposition. Furthermore, a different weight metric may be more applicable for application decomposition. For example, the number of bytes sent between device and server—and their direction—may be more representative of application behaviour than the number of flows.

References

[1] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, and L. E., "Address allocation for private internets [RFC1918]," *Internet Engineering Task Force (IETF)*, 1996.

[2] R. Venkateswaran, "Virtual private networks," *IEEE Potentials*, vol. 20, no. 1, pp. 11–15, 2001.

[3] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill, "How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation," in *Internet Measurement Conference*, p. 203–217, 2018.

[4] M. Zain ul Abideen, S. Saleem, and M. Ejaz, "VPN traffic detection in SSL-protected channel," *Security and Communication Networks*, vol. 2019, 2019.

[5] S. Webster, R. Lippmann, and M. Zissman, "Experience using active and passive mapping for network situational awareness," in *5th IEEE International Symposium on Network Computing and Applications (NCA'06)*, pp. 19–26, 2006.

[6] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen, "Network reconnaissance," *Network Security*, vol. 2008, no. 11, pp. 12–16, 2008.

[7] S. Mongkolluksamee, K. Fukuda, and P. Pongpaibool, "Counting NATted hosts by observing TCP/IP field behaviors," in *2012 IEEE International Conference on Communications (ICC)*, pp. 1265–1270, 2012.

[8] S. M. Bellovin, "A technique for counting NATted hosts," in *2nd ACM SIGCOMM Workshop on Internet Measurement*, pp. 267–272, 2002.

[9] A. Tekeoglu, N. Altiparmak, and A. S. Tosun, “Approximating the number of active nodes behind a NAT device,” in *20th IEEE International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, 2011.

[10] R. Mateless, H. Zlatokrilov, L. Orevi, M. Segal, and R. Moskovitch, “IPvest: Clustering the IP traffic of network entities hidden behind a single IP address using machine learning,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3647–3661, 2021.

[11] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, “Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[12] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, “Characterising network-connected devices using affiliation graphs,” in *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2020.

[13] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, “Operating system classification: A minimalist approach,” in *IEEE International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 143–150, 2020.

[14] T. Mikolov, W.-t. Yih, and G. Zweig, “Linguistic regularities in continuous space word representations,” in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human language Technologies*, pp. 746–751, 2013.

[15] G. Maier, F. Schneider, and A. Feldmann, “NAT usage in residential broadband networks,” Passive and Active Measurement, pp. 32–41, 2011.

[16] T. Komárek, M. Grill, and T. Pevný, “Passive NAT detection using HTTP access logs,” in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2006.

[17] A. S. Khatouni, L. Zhang, K. Aziz, I. Zincir, and N. Zincir-Heywood, “Exploring NAT detection and host identification using machine learning,” in *15th IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–8, 2019.

[18] Y. Meidan, V. Sachidananda, H. Peng, R. Sagron, Y. Elovici, and A. Shabtai, “A novel approach for detecting vulnerable IoT devices connected behind a home NAT,” *Computers & Security*, vol. 97, p. 101968, 2020.

[19] Y. Gokcen, V. A. Foroushani, and A. N. Z. Heywood, “Can we identify NAT behavior by analyzing traffic flows?,” in *2014 IEEE Security and Privacy Workshops*, pp. 132–139, 2014.

[20] S. Abt, C. Dietz, H. Baier, and S. Petrović, “Passive remote source NAT detection using behavior statistics derived from netflow,” Emerging Management Mechanisms for the Future Internet, pp. 148–159, 2013.

[21] A. D’Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, “A survey on big data for network traffic monitoring and analysis,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[22] Cisco, “Netflow version 9 flow-record format.” cisco.com, https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html. (accessed Oct. 17, 2021).

[23] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, “Enhancing situational awareness in encrypted networks using graph-based machine learning,” 2022. Under Submission.

[24] T. Pearce, A. Brintrup, M. Zaki, and A. Neely, “High-quality prediction intervals for deep learning: A distribution-free, ensembled approach,” in *PMLR International Conference on Machine Learning*, pp. 4075–4084, 2018.

[25] M. Jelassi, C. Ghazel, and L. A. Saïdane, “A survey on quality of service in cloud computing,” in *3rd International Conference on Frontiers of Signal Processing (ICFSP)*, pp. 63–67, 2017.

[26] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, “OS fingerprinting and tethering detection in mobile networks,” in *ACM Conference on Internet Measurement Conference*, pp. 173–180, 2014.

[27] S. Robertson, “Understanding inverse document frequency: on theoretical arguments for IDF,” *Journal of Documentation*, 2004.

[28] L. Bottou, “Stochastic gradient descent tricks,” in *Neural Networks: Tricks of the Trade*, pp. 421–436, 2012.

[29] B. Li, M. H. Gunes, G. Bebis, and J. Springer, “A supervised machine learning approach to classify host roles on line using sflow,” in *Workshop on High Performance and Programmable Networking (HPPN)*, pp. 53–60, 2013.

[30] R. Dhaya and M. Poongodi, “Detecting software vulnerabilities in android using static analysis,” in *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pp. 915–918, 2014.

[31] Q. Liang, X. Wu, and H. C. Lau, “Optimizing service systems based on application-level QoS,” *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 108–121, 2009.

Chapter 7

Conclusion

This chapter summarises the work conducted within this thesis. The key contributions of this thesis are enumerated in Section 7.1 and suggestions for future work are provided in Section 7.2. A closing statement is given in Section 7.3 concluding the thesis.

7.1 Contribution of Work

A total of nine papers were presented within this thesis. Each paper provided novel solutions to address the four key challenges of conducting network reconnaissance within modern networks: volume, variability, volatility, and visibility. The presented papers were combined into Chapters 3-6 based on the similarity of their research aims and contributions. The contribution of each chapter is summarised as follows:

Chapter 2 evaluated 56 passive network reconnaissance solutions that were identified in related literature. No surveyed solution was found to satisfy all four criteria for widespread deployment on realistic network conditions:

- Criterion 1 (CN1) - Encryption Independence
- Criterion 2 (CN2) - Universal Minimum Feature Set
- Criterion 3 (CN3) - Real-World Deployment
- Criterion 4 (CN4) - Long-Term Deployment

We addressed this contribution gap through the development of novel graph-based network reconnaissance solutions. In Chapters 4-6, we provided the first known passive network reconnaissance solutions that address all four defined criteria (CN1-4).

In Chapter 3, two papers were provided on the use of conventional machine learning (ML) for network reconnaissance. An empirical justification was first provided on the prevalent use of random forest (RF) and convolutional neural networks (CNNs) within the network reconnaissance domain. A novel CNN architecture—*Segmented-CNN*—was then designed to exploit the unique structural properties of the TCP/IP protocol stack. The Segmented-CNN was shown to reduce the training time of a CNN classifier and improve robustness to evasive malicious behaviour. Furthermore, we discovered the proclivity of neural network architectures to overfit when evaluated on full packet captures. This discovery prompted the investigation into a graph-based point-of-analysis (PoA) for network reconnaissance.

In Chapter 4, four papers were provided on the use of a bipartite graph-based PoA for conducting passive network reconnaissance. Our designed bipartite PoA provides the first comprehensive framework for conducting passive network reconnaissance that relies only on the source and destination IP address fields. This widely available feature set was exploited to produce network reconnaissance solutions that are scalable, independent of encryption (CN1), and deployable across diverse Internet (TCP/IP) networks (CN2). Furthermore, we showed that the bipartite PoA is effective for long-term deployment (CN4) on real-world networks (CN3) through the analysis of a university campus network over a six-month observation.

In Chapter 5, two papers were provided on the use of graph-based ML on the bipartite PoA. In particular, we designed bipartite graph embeddings (BGE); the first graph embedding technique that enables the real-time analysis of a large TCP/IP network. We show that BGE remains effective under partial network observation and efficiently scales for the analysis of networks containing hundreds of thousands of devices. Furthermore, we show that the embeddings produced by BGE can be reused to satisfy distinct network reconnaissance objectives (e.g., device and application characterisation). BGE thus provides a comprehensive methodology for conducting passive network reconnaissance.

In Chapter 6, a single paper was provided on the use of graph-based machine learning for the analysis of private networks. In particular, we designed PiPiN; an extension of BGE that utilises Prediction intervals to confidently acquire situational awareness behind a Private Network. PiPiN was shown to accurately estimate the number of devices within private networks that consist of up to 500 devices. Furthermore, we show that PiPiN can estimate the composition of device manufacturers, operating systems, and applications that are used within a private network. PiPiN therefore provides a network operator with crucial insight into the characteristics of a private network.

7.2 Future Work

We acknowledge that all research has limitations. In this section, we highlight opportunities for the contributions of this thesis to be extended in future work. The areas of this thesis that require future contribution (FC) are identified as follows:

FC1 **Peer-to-peer communications:** This thesis focused on the representation of a TCP/IP network within a bipartite graph as first posed in Chapter 4. The edges in a bipartite graph exist only between vertices of two distinct sets. The two distinct sets were used to represent 1) the devices internal to a TCP/IP network and 2) the Internet services used by the observed devices. This property exploited the pervasive client-server architecture of TCP/IP communications.

A peer-to-peer (P2P) architecture cannot be represented by a bipartite graph. This limitation would result in information loss when representing a TCP/IP network within the bipartite graph-based PoA. For example, internal scanning—an early warning sign of a compromised network—would not be detected as it results in direct communication between devices internal to the network.

FC2 **Fine-grain characterisation:** The Internet services used by a device were shown to provide sufficient insight for intrusion detection, device characterisation, and application characterisation. For instance, the operating system (OS) of device was shown to be accurately predicted by the device's affiliated Internet services. Additional information would be required to develop techniques capable of fine-grain characterisation. Fine-grain characterisation (e.g., identifying the version of a device's OS) would foster a deeper understanding of the network under analysis.

FC3 **Active network reconnaissance:** This thesis focused solely on the use of passive network reconnaissance. Active network reconnaissance (i.e., directly interacting with the observed network) would provide auxiliary information that may not be obtainable through passive network reconnaissance alone. For example, passive network reconnaissance cannot resolve any insight into a device if the device is inactive on the network.

FC4 **Geo-restricted analysis:** The Internet services used by a device are dependent on the device's location. For example, a device located in Australia will prioritise Australian-based Internet services to reduce the latency of its communications. This property necessitates labelled (*seed*) devices to be sourced from the network under analysis or a network located in the same location. This property may limit the insight produced by the techniques developed within this thesis if appropriate seed devices cannot be sourced.

FC5 **Quantitative Comparison:** In Chapter 2, a qualitative metric (CN1-4) was defined to compare the techniques provided within this thesis to those presented in literature. This qualitative metric highlighted that the techniques presented in literature are limited in their widespread applicability (CN2) due to their use of deep packet inspection (DPI) or large, heterogeneous feature sets. In Chapters 4-6, we designed novel graph-based network reconnaissance techniques to address this limitation.

A quantitative comparison was not provided in this thesis. A quantitative comparison can only be provided on the condition that all techniques are evaluated on the same dataset. This condition was not satisfied as the techniques developed in literature could not be deployed as they required features that were unavailable on the university network under analysis.

The remainder of this section defines four avenues of future work to extend the listed areas of future contribution (FC). In Section 7.2.1, we propose alternative graph-based PoAs for addressing FC1-2. In Section 7.2.2, we propose two methods for incorporating active network reconnaissance into the solutions provided within this thesis (FC3). In Section 7.2.3, we propose a method of utilising FC4 to extend the solutions developed within this thesis for user identification. In Section 7.2.4, we define the requirements to quantitatively compare the techniques developed in this thesis to those provided in literature (FC5).

7.2.1 Extended Graph-Based Analysis for Improved Network Situational Awareness

In Chapter 4, a bipartite PoA was designed and evaluated. This bipartite PoA is limited in its representation of P2P communications and for performing fine-grain characterisations.

We provide two avenues of future work to address the limitations of the bipartite PoA:

1. **heterogeneous graph-based PoA:** A heterogeneous graph can be used to encode numerous characteristics of a TCP/IP network directly into the graph's structure. These characteristics (e.g., the ports used by a device or which devices are used by multiple users) would provide additional insight to improve a network operator's situational awareness. In addition, a heterogeneous graph can represent the inter- and intra- communications of a TCP/IP network; thus enabling the representation of P2P communications.
2. **Time series graph-based PoA:** A time series graph can be used to provide insight into the longitudinal behaviour of a TCP/IP network. In particular, a time series graph could be used to represent the pattern-of-life of a device or its user; thus enabling techniques to perform a fine-grain characterisation. For example, distinct versions of an OS can be distinguished by the periodicity in which they communicate with Internet services.

7.2.2 Enhancing Bipartite Graph Embeddings Through Active Network Reconnaissance

In Chapter 5, we designed and evaluated bipartite graph embeddings (BGE). BGE can be extended through the use of active network reconnaissance to supplement the situational awareness that can be achieved through passive network reconnaissance alone.

We provide two avenues of future work for enhancing BGE through active network reconnaissance:

1. **Active resolution of seed devices:** BGE is an unsupervised ML technique that identifies clusters of devices with similar behavioural characteristics. A small set of labelled (*seed*) devices are required to label their respective cluster. Active reconnaissance could be used to automatically produce seed devices on a TCP/IP network by probing the set of devices central to each identified cluster. This extension provides the benefits of active reconnaissance while significantly reducing the number of devices that are required to be actively probed. This benefit would vastly reduce the disruption to a network caused by pervasive active network reconnaissance.

2. **Active learning and reconnaissance:** Active learning is a ML optimisation strategy which interactively queries a knowledge source (e.g., a domain expert) when there is uncertainty in a decision or prediction. Active reconnaissance could be utilised as a knowledge source that probes a TCP/IP network when there is uncertainty in the embeddings produced by BGE. For example, active reconnaissance could be used to resolve a device's characteristics if the device is found to belong to multiple known clusters. This extension provides the benefits of active reconnaissance while limiting its use to only the devices that cannot be characterised through passive network reconnaissance.

7.2.3 Bipartite Point-of-Analysis for User Identification

In Chapter 4, we evaluate the bipartite PoA for intrusion detection and device characterisation. The bipartite PoA was inspired by methods for user identification in the social network theory domain. It is therefore expected that the bipartite PoA—as designed within this thesis—can provide useful insight into user identification in a TCP/IP network. That is, the characteristics of a user (e.g., age, location, and gender) could be inferred from a user's affiliated Internet services.

We provide two avenues of future work for the utilisation of the bipartite PoA for user identification:

1. **User fingerprinting:** User fingerprinting aims to identify a specific user based on their observed behaviour on a TCP/IP network. User fingerprinting has been motivated by sociology studies, market research, and to detect the leakage of personal identifiable information (PII). The bipartite PoA is expected to be beneficial for user fingerprinting by identifying a user's characteristics based on their affiliated Internet services. For example, identifying the location of a user based on their access of geo-restricted Internet services.
2. **User role identification:** User role identification aims to identify the role of a user (e.g., administrator, employee, or student). The motivation of user role identification is to detect anomalous user behaviour within a TCP/IP network; that is, a user acting outside their assigned role. The bipartite PoA is expected to enable the detection of user roles through clustering users by their affiliated Internet services. The bipartite PoA would require modification to include services internal to the network. This modification is necessary for user role identification as it would assist in separating network administrators (with unrestricted access to internal services) from everyday users (with restricted access to internal services).

7.2.4 Quantitative Comparison

Full packet captures (*pcaps*) are required to compare the techniques developed within this thesis to those provided in literature. Pcaps contain a full record of the network traffic that has passed through an observation point and thus provide all features that are required for the deployment of diverse passive network reconnaissance solutions.

The techniques developed within this thesis were primarily evaluated on the University of Adelaide's (UofA's) operational network. Pcaps could not be obtained from the evaluated network as they can contain confidential information. The UofA's network cannot be used to perform a quantitative comparison as the techniques provided in literature could not be deployed on the network. In addition, statistical hypothesis tests should be conducted to ensure the comparison between techniques is statistically significant.

A research agreement with a network operator is required to capture pcaps and thus perform a quantitative comparison on a real-world (CN3) operational network. The research agreement must uphold ethical and privacy considerations to ensure that the pcaps are processed and stored securely. We provide open access to the code base developed for BGE[1] to enable a quantitative comparison to be conducted in future work.

7.3 Closing Statement

The techniques designed and evaluated within this thesis provide novel solutions for two crucial aims of network reconnaissance—device characterisation and intrusion detection. This thesis addressed known limitations in current literature by providing network reconnaissance techniques that are scalable, independent of encryption, and deployable across

[1]`https://github.com/MillarK-UofA/bipartite_graph_embeddings`

diverse TCP/IP networks. We provide these techniques for the development of unique cybersecurity solutions to both prevent and mitigate the impact of future cyber-attacks.

Bibliography

[1] A. Gelnaw, "You can't secure what you can't see." bitsight.com, `https://www.bitsight.com/blog/you-cant-secure-what-you-cant-see`. (accessed Apr. 26, 2022).

[2] C. Harber, "You can't defend what you can't see: Why visibility is critical for improving cyber defense." securityweek.com, `https://www.securityweek.com/you-cant-defend-what-you-cant-see-why-visibility-critical-improving-cyber-defense`. (accessed Apr. 26, 2022).

[3] P. Quade, "You can't protect what you can't see." csoonline.com, `https://www.csoonline.com/article/3256211/you-cant-protect-what-you-cant-see.html`. (accessed Apr. 26, 2022).

[4] J. Faircloth, *Penetration tester's open source toolkit.* Syngress, 2016.

[5] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen, "Network reconnaissance," *Network Security*, vol. 2008, no. 11, pp. 12–16, 2008.

[6] P. A. Frangoudis, L. Yala, and A. Ksentini, "AWESoME: Big data for automatic web service management in SDN," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 13–26, 2018.

[7] N. Shah, "Industry trends: The challenges of inspecting encrypted network traffic." fortinet.com, `https://www.fortinet.com/blog/industry-trends/keeping-up-with-performance-demands-of-encrypted-web-traffic`. (accessed Apr. 26, 2022).

[8] Google, "Google transparency report: HTTPS encryption on the web." transparencyreport.google.com, `https://transparencyreport.google.com/https/overview?hl=en`, 2022. (accessed Apr. 26, 2022).

[9] G. Aceto, G. Bovenzi, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Characterization and prediction of mobile-app traffic using markov modeling," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 907–925, 2021.

[10] M. Conti, Q. Li, A. Maragno, and R. Spolaor, "The dark side (-channel) of mobile devices: A survey on network traffic analysis," *arXiv preprint arXiv:1708.03766*, 2018.

[11] B. Schneier, "Click here to kill everybody: Security and survival in a hyper-connected world," p. 54, 2018.

[12] Australian Cyber Security Centre (ACSC), "ASCS annual cyber threat report: July 2020 to july 2021." cyber.gov.au, `https://www.cyber.gov.au/sites/default/files/2021-09/ACSC\%20Annual\%20Cyber\%20Threat\%20Report\%20-\%202020-2021.pdf`. (accessed Apr. 26, 2022).

[13] Lowy Institute, "Poll 2021 - security and defence." poll.lowyinstitute.org, `https://poll.lowyinstitute.org/themes/security-and-defence`. (accessed: Jan. 13, 2022).

[14] Australian Cyber Security Centre (ACSC), "ACSC annual cyber threat report: July 2019 to july 2020." cyber.gov.au, `https://www.cyber.gov.au/sites/default/files/2020-09/ACSC-Annual-Cyber-Threat-Report-2019-20.pdf`. (accessed Apr. 26, 2022).

[15] CISCO, "What is cybersecurity?." cisco.com, `https://www.cisco.com/c/en_au/products/security/what-is-cybersecurity.html`. (accessed Apr. 26, 2022).

[16] National Institute of Standards and Technology, "Framework for improving critical infrastructure cybersecurity." nvlpubs.nist.gov, `https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf`. (accessed Apr. 26, 2022).

[17] R. J. Barnett and B. Irwin, “Towards a taxonomy of network scanning techniques,” in *2008 South African Institute of Computer Scientists and Information Technologists (SAICSIT’08)*, p. 1–7, 2008.

[18] B. AsSadhan, J. M. Moura, D. Lapsley, C. Jones, and W. T. Strayer, “Detecting botnets using command and control traffic,” in *8th IEEE International Symposium on Network Computing and Applications*, pp. 156–162, 2009.

[19] Australian National University (ANU), “Incident report: On the breach of the australian national university’s administrative systems.” imagedepot.anu.edu.au, `https://imagedepot.anu.edu.au/scapa/Website/SCAPA190209_Public_report_web_2.pdf`. (accessed Apr. 26, 2022).

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, 2017.

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, “Language models are few-shot learners,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[22] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[23] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, “A survey of machine learning for big data processing,” *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.

[24] E. Bou-Harb, M. Debbabi, and C. Assi, “Cyber scanning: A comprehensive survey,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1496–1519, 2014.

[25] S. Webster, R. Lippmann, and M. Zissman, “Experience using active and passive mapping for network situational awareness,” in *5th IEEE International Symposium on Network Computing and Applications (NCA’06)*, pp. 19–26, 2006.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 1–21, 2020.

[27] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 4, pp. 56–69, 2007.

[28] P. V. Amoli, T. Hamalainen, G. David, M. Zolotukhin, and M. Mirzamohammad, "Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets," *International Journal of Digital Content Technology and its Applications (JDCTA)*, vol. 10, no. 2, pp. 1–13, 2016.

[29] L. Bilge and T. Dumitraş, "Before we knew it: an empirical study of zero-day attacks in the real world," in *2012 ACM Conference on Computer and Communications Security*, pp. 833–844, 2012.

[30] M. Sarhan, S. Layeghy, M. Gallagher, and M. Portmann, "From zero-shot machine learning to zero-day attack detection," *arXiv preprint arXiv:2109.14868*, 2021.

[31] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *International Conference on Information Networking (ICOIN)*, pp. 712–717, 2017.

[32] A. Arora and S. K. Peddoju, "Minimizing network traffic features for android mobile malware detection," in *18th International Conference on Distributed Computing and Networking*, pp. 1–10, 2017.

[33] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.

[34] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[35] D. Arora, K. F. Li, and A. Loffler, "Big data analytics for classification of network enabled devices," in *Advanced Information Networking and Applications Workshops (WAINA)*, pp. 708–713, 2016.

[36] Q. Chen and R. A. Bridges, "Automated behavioral analysis of malware: A case study of wannacry ransomware," in *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 454–460, 2017.

[37] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "PiPiN: Acquiring situational awareness behind private networks with confidence," unpublished paper, School of Electrical and Electronic Engineering, The University of Adelaide, 2022.

[38] K. Downer and M. Bhattacharya, "BYOD security: A new business challenge," in *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp. 1128–1133, 2015.

[39] N. Ammar, L. Noirie, and S. Tixeuil, "Autonomous identification of IoT device types based on a supervised classification," in *International Conference on Communications (ICC)*, pp. 1–6, 2020.

[40] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, "OS fingerprinting and tethering detection in mobile networks," in *Conference on Internet Measurement*, pp. 173–180, 2014.

[41] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2017.

[42] M. Lastovicka, T. Jirsik, P. Celeda, S. Spacek, and D. Filakovsky, "Passive OS fingerprinting methods in the jungle of wireless networks," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2018.

[43] S. Mongkolluksamee, K. Fukuda, and P. Pongpaibool, "Counting NATted hosts by observing TCP/IP field behaviors," in *IEEE International Conference on Communications (ICC)*, pp. 1265–1270, 2012.

[44] S. M. Bellovin, “A technique for counting NATted hosts,” in *2nd ACM SIGCOMM Workshop on Internet Measurement*, pp. 267–272, 2002.

[45] A. Tekeoglu, N. Altiparmak, and A. S. Tosun, “Approximating the number of active nodes behind a NAT device,” in *20th IEEE International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, 2011.

[46] R. Mateless, H. Zlatokrilov, L. Orevi, M. Segal, and R. Moskovitch, “IPvest: Clustering the IP traffic of network entities hidden behind a single IP address using machine learning,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3647–3661, 2021.

[47] M. Willett, “Lessons of the solarwinds hack,” *Survival*, vol. 63, no. 2, pp. 7–26, 2021.

[48] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, “Robust smartphone app identification via encrypted network traffic analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2018.

[49] H. K. Lim, J. B. Kim, J. S. Heo, K. Kim, Y. G. Hong, and Y. H. Han, “Packet-based network traffic classification using deep learning,” in *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 046–051, 2019.

[50] S. Dong, “Multi class SVM algorithm with active learning for network traffic classification,” *Expert Systems with Applications*, vol. 176, 2021.

[51] J. Li, H. Zhou, S. Wu, X. Luo, T. Wang, X. Zhan, and X. Ma, “FOAP: Fine-grained open-world android app fingerprinting,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[52] K. Park and H. Kim, “Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis,” in *International Workshop on Information Security Applications*, pp. 254–265, 2015.

[53] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, “Investigating user privacy in android ad libraries,” in *Workshop on Mobile Security Technologies (MoST)*, vol. 10, pp. 195–197, 2012.

[54] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: Smartphone fingerprinting via application behaviour," in *6th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 7–12, 2013.

[55] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling, "Fingerprinting mobile devices using personalized configurations," *Privacy Enhancing Technologies*, vol. 2016, no. 1, pp. 4–19, 2016.

[56] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "Recon: Revealing and controlling PII leaks in mobile network traffic," in *14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 361–374, 2016.

[57] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, 2009.

[58] G. E. Bartlett, *Network reconnaissance using blind techniques.* Ph.D. dissertation, University of Southern California, 2010.

[59] G. E. Bartlett, J. Heidemann, and C. Papadopoulos, "Understanding passive and active service discovery," in *7th ACM SIGCOMM Conference on Internet measurement*, pp. 57–70, 2007.

[60] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach.* Pearson Education Limited, 6th ed., 2013.

[61] J. Martin, E. Rye, and R. Beverly, "Decomposition of MAC address structure for granular device inference," in *32nd ACM Annual Conference on Computer Security Applications*, pp. 78–88, 2016.

[62] M. Zalewski, "p0fv3." lcamtuf.coredump.cx, `https://lcamtuf.coredump.cx/p0f3`. (accessed Mar. 7, 2022).

[63] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[64] P. A. B. Claise, B. Trammell, "Specification of the ip flow information export (IPFIX) protocol for the exchange of flow information [RFC7011]," *Internet Engineering Task Force (IETF)*, 2013.

[65] M. Zhan, Y. Li, G. Yu, B. Li, and W. Wang, "Detecting DNS over HTTPS based data exfiltration," *Computer Networks*, vol. 209, 2022.

[66] J. Kampeas, A. Cohen, and O. Gurewitz, "Traffic classification based on zero-length packets," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1049–1062, 2018.

[67] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: A machine learning approach for IoT device identification based on network traffic analysis," in *ACM Symposium on Applied Computing*, pp. 506–509, 2017.

[68] T. Radivilova, L. Kirichenko, D. Ageyev, M. Tawalbeh, and V. Bulakh, "Decrypting SSL/TLS traffic for hidden threats detection," in *9th IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 143–146, 2018.

[69] A. Alshamsi and T. Saito, "A technical comparison of IPSec and SSL," in *19th International Conference on Advanced Information Networking and Applications (AINA'05)*, pp. 395–398, 2005.

[70] V. Korhonen, "Future after OpenVPN and IPsec," 2019. M.S. Thesis, School of Computing and Electrical Engineering, Tampere University, 2019.

[71] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[72] Cisco, "Netflow version 9 flow-record format." cisco.com, `https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html`. (accessed Oct. 17, 2021).

[73] Internet Assigned Numbers Authority (IANA), "IP flow information export (IPFIX) entities." iana.org, `https://www.iana.org/assignments/ipfix/ipfix.xhtml`. (accessed Oct. 17, 2021).

[74] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related," in *2nd International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 407–414, 2016.

[75] Internet Assigned Numbers Authority (IANA), "Service name and transport protocol port number registry." iana.org, `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`. (accessed Jun. 21, 2022).

[76] J. Zhao, X. Jing, Z. Yan, and W. Pedrycz, "Network traffic classification for data fusion: A survey," *Information Fusion*, vol. 72, pp. 22–47, 2021.

[77] ntop, "nDPI: Open and extensible LGPLv3 deep packet inspection library." ntop.org, `https://www.ntop.org/products/deep-packet-inspection/ndpi/`. (accessed Oct. 4, 2021).

[78] statista, "Number of available applications in the Google Play Store from December 2009 to March 2022." statista.com, `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`. (accessed Oct. 4, 2021).

[79] B. Schneier, "The internet of things will upend our industry," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 108–108, 2017.

[80] Y. Kazato, Y. Nakagawa, and Y. Nakatani, "Improving maliciousness estimation of indicator of compromise using graph convolutional networks," in *17th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–7, 2020.

[81] R. Kozik, M. Pawlicki, and M. Choraś, "Cost-sensitive distributed machine learning for netflow-based botnet activity detection," *Security and Communication Networks*, 2018.

[82] A. S. Uluagac, S. V. Radhakrishnan, C. Corbett, A. Baca, and R. Beyah, "A passive technique for fingerprinting wireless devices with wired-side observations," in *IEEE Conference on Communications and Network Security (CNS)*, pp. 305–313, 2013.

[83] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.

[84] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *2014 IEEE Conference on Communications and Network Security*, pp. 247–255, 2014.

[85] F. Zola, L. Segurola-Gil, J. Bruse, M. Galar, and R. Orduna-Urrutia, "Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing," *Computers & Security*, vol. 115, 2022.

[86] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[87] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[88] A. Caciularu, N. Raviv, T. Raviv, J. Goldberger, and Y. Be'ery, "perm2vec: Attentive graph permutation selection for decoding of error correction codes," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 79–88, 2021.

[89] T. Matsunaka, A. Yamada, and A. Kubota, "Passive OS fingerprinting by DNS traffic analysis," in *27th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 243–250, 2013.

[90] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No NAT'd user left behind: Fingerprinting users behind NAT from NetFlow records alone," in *34th IEEE International Conference on Distributed Computing Systems*, pp. 218–227, 2014.

[91] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, and J. Li, "A first look at android malware traffic in first few minutes," in *IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 206–213, 2015.

[92] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE Conference on Communications and Network Security (CNS)*, pp. 433–441, 2015.

[93] N. Ruffing, Y. Zhu, R. Libertini, Y. Guan, and R. Bettati, "Smartphone reconnaissance: Operating system identification," in *13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1086–1091, 2016.

[94] T. Komárek, M. Grill, and T. Pevný, "Passive NAT detection using HTTP access logs," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, 2016.

[95] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized IoT devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[96] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2Img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks," in *IEEE International Conference on Big Data*, pp. 1271–1276, 2017.

[97] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 43–48, 2017.

[98] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic," *arXiv preprint arXiv:1705.06805*, 2017.

[99] S. Bagui, X. Fang, E. Kalaimannan, S. C. Bagui, and J. Sheehan, "Comparison of machine-learning algorithms for classification of vpn network traffic flow using time-related features," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 108–126, 2017.

[100] Y. Shang, S. Yang, and W. Wang, "Botnet detection with hybrid analysis on flow based and graph based features of network traffic," in *International Conference on Cloud Computing and Security*, pp. 612–621, 2018.

[101] J. S. Atkinson, J. E. Mitchell, M. Rio, and G. Matich, "Your WiFi is leaking: What do your mobile apps gossip about you?," *Future Generation Computer Systems*, vol. 80, pp. 546–557, 2018.

[102] L. Vu, H. V. Thuy, Q. U. Nguyen, T. N. Ngoc, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Time series analysis for encrypted traffic classification: A deep learning approach," in *18th IEEE International Symposium on Communications and Information Technologies (ISCIT)*, pp. 121–126, 2018.

[103] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2018.

[104] A. S. Khatouni, L. Zhang, K. Aziz, I. Zincir, and N. Zincir-Heywood, "Exploring NAT detection and host identification using machine learning," in *15th IEEE International Conference on Network and Service Management (CNSM)*, pp. 1–8, 2019.

[105] J. Zhang, F. Li, F. Ye, and H. Wu, "Autonomous unknown-application filtering and labeling for DL-based traffic classifier update," in *IEEE Conference on Computer Communications (INFOCOM)*, pp. 397–405, 2020.

[106] X. Tong, X. Tan, L. Chen, J. Yang, and Q. Zheng, "BFSN: A novel method of encrypted traffic classification based on bidirectional flow sequence network," in *3rd IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pp. 160–165, 2020.

[107] Y. He and W. Li, "Image-based encrypted traffic classification with convolution neural networks," in *5th IEEE International Conference on Data Science in Cyberspace (DSC)*, pp. 271–278, 2020.

[108] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even

encrypted!," in *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 207–218, 2020.

[109] R. Nigmatullin, A. Ivchenko, and S. Dorokhin, "Differentiation of sliding rescaled ranges: New approach to encrypted and VPN traffic detection," in *International Conference Engineering and Telecommunication (En&T)*, pp. 1–5, 2020.

[110] Y. Meidan, V. Sachidananda, H. Peng, R. Sagron, Y. Elovici, and A. Shabtai, "A novel approach for detecting vulnerable IoT devices connected behind a home NAT," *Computers & Security*, vol. 97, 2020.

[111] Z. Chen, G. Cheng, B. Jiang, S. Tang, S. Guo, and Y. Zhou, "Length matters: Fast internet encrypted traffic service classification based on multi-pdu lengths," in *16th IEEE International Conference on Mobility, Sensing and Networking (MSN)*, pp. 531–538, 2020.

[112] A. S. Iliyasu and H. Deng, "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks," *IEEE Access*, vol. 8, pp. 118–126, 2020.

[113] M. Bahaa, A. Aboulmagd, K. Adel, H. Fawzy, and N. Abdelbaki, "nnDPI: A novel deep packet inspection technique using word embedding, convolutional and recurrent neural networks," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pp. 165–170, 2020.

[114] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2020.

[115] W. Satrabhandhu and S. Tritilanunt, "Encrypted traffic characterization using none zero payload and payload ratio characteristics," in *25th IEEE International Computer Science and Engineering Conference (ICSEC)*, pp. 63–69, 2021.

[116] S. Zhang, Z. Wang, J. Yang, D. Bai, F. Li, Z. Li, J. Wu, and X. Liu, "Unsupervised IoT fingerprinting method via variational auto-encoder and k-means," in *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2021.

[117] W. Maonan, Z. Kangfeng, X. Ning, Y. Yanqing, and W. Xiujuan, "CENTIME: A direct comprehensive traffic features extraction for encrypted traffic classification," in *6th IEEE International Conference on Computer and Communication Systems (ICCCS)*, pp. 490–498, 2021.

[118] Q. Ma, W. Huang, Y. Jin, and J. Mao, "Encrypted traffic classification based on traffic reconstruction," in *4th IEEE International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pp. 572–576, 2021.

[119] A. Parchekani, S. N. Naghadeh, and V. Shah-Mansouri, "Classification of traffic using neural networks by rejecting: a novel approach in classifying VPN traffic," *arXiv preprint arXiv:2001.03665*, 2020.

[120] M. Lu, B. Zhou, Z. Bu, K. Zhang, and Z. Ling, "Compressed network in network models for traffic classification," in *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2021.

[121] S. Izadi, M. Ahmadi, and A. Rajabzadeh, "Network traffic classification using deep learning networks and bayesian data fusion," *Journal of Network and Systems Management*, vol. 30, no. 2, pp. 1–21, 2022.

[122] W. Zheng, J. Zhong, Q. Zhang, and G. Zhao, "MTT: an efficient model for encrypted network traffic classification using multi-task transformer," *Applied Intelligence*, pp. 1–16, 2022.

[123] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu, "Identification of encrypted traffic through attention mechanism based long short term memory," *IEEE Transactions on Big Data*, 2022.

[124] F. Zaki, F. Afifi, S. Abd Razak, A. Gani, and N. B. Anuar, "GRAIN: Granular multi-label encrypted traffic classification using classifier chain," *Computer Networks*, 2022.

[125] A. Telikani, A. H. Gandomi, K.-K. R. Choo, and J. Shen, "A cost-sensitive deep learning-based approach for network traffic classification," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 661–670, 2021.

[126] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Using convolutional neural networks for classifying malicious network traffic," in *Deep Learning Applications for Cyber Security*, pp. 103–126, 2019.

[127] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Characterising network-connected devices using affiliation graphs," in *IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2020.

[128] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Clustering network-connected devices using affiliation graphs," in *IEEE International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 1–6, 2021.

[129] A. Cheng and K. Millar, "Detecting data exfiltration using seeds based graph clustering," in *2022 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, 2022. Under Submission.

[130] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Operating system classification: A minimalist approach," in *IEEE International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 143–150, 2020.

[131] K. Millar, L. Simpson, A. Cheng, H. G. Chew, and C.-C. Lim, "Detecting botnet victims through graph-based machine learning," in *IEEE International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 1–6, 2021.

[132] K. Millar, A. Cheng, H. G. Chew, and C.-C. Lim, "Enhancing situational awareness in encrypted networks using graph-based machine learning," unpublished paper, School of Electrical and Electronic Engineering, The University of Adelaide, 2022.

[133] R. L. Breiger, "The duality of persons and groups," *Social Forces*, vol. 53, no. 2, pp. 181–190, 1974.

[134] J. B. Merrill, "Liberal, moderate or conservative? see how Facebook labels you." nytimes.com, `https://www.nytimes.com/2016/08/24/us/politics/facebook-ads-politics.html`. (accessed Jul. 5, 2022).

[135] E. Schaeffer, "Stochastic local clustering for massive graphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 354–360, Springer, 2005.