NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CLASSIFYING TCP NETWORK TRAFFIC FLOWS VIA TRAFFIC INTERACTION GRAPHS AND MACHINE LEARNING

by

Matthew N. Straughn

September 2023

Thesis Advisor: Armon C. Barton
Second Reader: Gurminder Singh

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2023	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
CLASSIFYING TCP NETWORK TRAFFIC FLOWS VIA TRAFFIC INTERACTION GRAPHS AND MACHINE LEARNING

5. FUNDING NUMBERS

6. AUTHOR(S) Matthew N. Straughn

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Detecting malicious traffic on networks is a critical problem facing the Department of Defense. In this thesis we utilize cutting edge machine learning techniques to detect malicious network traffic. We begin with two real-world datasets. First, real internet traffic collected on the NPS Enterprise Research Network, and second, the IoT23 dataset consisting of internet of things (IoT) devices that have been infected with malware. We convert raw packet captures into TCP streams using the 5-tuple definition from RFC6146. We then apply the traffic interaction graph (TIG) framework to these flows to capture burst patterns among signed packet lengths. Finally, we train these flows on a random forest classifier (RFC), a simple convolutional neural network (CNN), and a graph convolutional network (GCN). Additionally, we simulate various attack levels by combining the two datasets at various levels (99% NPS to 1% IoT, 99–5, 90–10, and IoT23 only). In each of these models we get exceptional results in accuracy, precision, and recall. This work specifically provides a proof of concept for using the TIG framework and graph neural networks to classify TCP flows. Future work should explore model enhancement, data enrichment, or stream-lining the entire process into a real-time software package.

14. SUBJECT TERMS
TCP, network traffic, flows, streams, graph neural network, graph convolutional network, GCN, convolutional neural network, CNN, random forest classifier, RFC, traffic interaction graph, TIG

15. NUMBER OF PAGES
79

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

CLASSIFYING TCP NETWORK TRAFFIC FLOWS VIA TRAFFIC INTERACTION GRAPHS AND MACHINE LEARNING

Matthew N. Straughn
Lieutenant, United States Navy
BS, University of New Mexico, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2023

Approved by: Armon C. Barton
Advisor

Gurminder Singh
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Detecting malicious traffic on networks is a critical problem facing the Department of Defense. In this thesis we utilize cutting edge machine learning techniques to detect malicious network traffic. We begin with two real-world datasets. First, real internet traffic collected on the NPS Enterprise Research Network, and second, the IoT23 dataset consisting of internet of things (IoT) devices that have been infected with malware. We convert raw packet captures into TCP streams using the 5-tuple definition from RFC6146. We then apply the traffic interaction graph (TIG) framework to these flows to capture burst patterns among signed packet lengths. Finally, we train these flows on a random forest classifier (RFC), a simple convolutional neural network (CNN), and a graph convolutional network (GCN). Additionally, we simulate various attack levels by combining the two datasets at various levels (99% NPS to 1% IoT, 99–5, 90–10, and IoT23 only). In each of these models we get exceptional results in accuracy, precision, and recall. This work specifically provides a proof of concept for using the TIG framework and graph neural networks to classify TCP flows. Future work should explore model enhancement, data enrichment, or stream-lining the entire process into a real-time software package.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Research Questions . . . 2

1.3 Scope . . . 2

1.4 Contributions . . . 2

1.5 Organization . . . 3

2 Prerequisites **5**

2.1 Overview . . . 5

2.2 Machine Learning . . . 5

2.3 Graph Theory . . . 10

2.4 Graph Neural Networks . . . 19

2.5 Network Traffic Analysis . . . 25

3 Methodology **29**

3.1 Python Implementation . . . 29

3.2 Data Transformation . . . 30

3.3 Machine Learning Models . . . 34

4 Experimental Setup **37**

4.1 Data . . . 37

4.2 Data Exploration and Visualization . . . 39

4.3 Ratios . . . 44

4.4 Results . . . 45

5 Conclusion **53**

5.1 Summary and Discussion . . . 53

5.2 Future Work . . . 54

List of References 57

Initial Distribution List 61

List of Figures

Figure 2.1 Decision Tree Example 6
Figure 2.2 Introductory Graph Example 11
Figure 2.3 Simple Connected Graph 12
Figure 2.4 Graph with highlighted edges 13
Figure 2.5 Directed Graph Example 14
Figure 2.6 Graph where neighbors are highlighted 14
Figure 2.7 Tree Example 15
Figure 2.8 Subgraph Example 16
Figure 2.9 Graph Isomorphism Example 17
Figure 2.10 Adjacency Matrix of a Graph 18
Figure 2.11 Laplacian Matrix of a Graph 19

Figure 3.1 Client Server Interaction 31
Figure 3.2 Example Traffic Interaction Graph. 33

Figure 4.1 NPS ERN Data Collection Points. 38
Figure 4.2 NPS ERN Flow Length vs. (log) Count. 40
Figure 4.3 IoT-23 Flow Length (all data) vs. (log) Count. 41
Figure 4.4 Comparison of IoT-23 malicious and benign flow lengths 42
Figure 4.5 Histogram of Signed Packet Length (Packet 25) of IoT-23 43
Figure 4.6 Example real TIG graphs 44

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 2.1 Confusion Matrix 9

Table 2.2 Confusion Matrix Illustration 9

Table 3.1 Convolutional Neural Network Model Architecture. 35

Table 4.1 RFC Split Metrics 46

Table 4.2 RFC IoT-23 Metrics 46

Table 4.3 CNN Split Metrics 47

Table 4.4 CNN IoT-23 Metrics 47

Table 4.5 GCN Split Metrics 48

Table 4.6 GCN IoT-23 Metrics 49

Table 4.7 99-1 Model Comparison 50

Table 4.8 95-5 Model Comparison 50

Table 4.9 90-10 Model Comparison 50

Table 4.10 IoT-23 Model Comparison 51

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

APT advanced persistent threat

API application programming interface

CNN convolutional neural network

DGL deep graph library

DL deep learning

DOD Department of Defense

DON Department of the Navy

DAG directed acyclic graph

ERN Enterprise Research Network

FN false negative

FP false positive

GCN graph convolutional network

GNN graph neural network

GRL graph representation learning

IoT internet of things

ML machine learning

NPS Naval Postgraduate School

OGB open graph benchmark

PCAP packet capture

PyG pytorch geometric

RFC random forest classifier

RNN recurrent neural network

TIG traffic interaction graph

TCP transmission control protocol

TCP/IP transmission control protocol/internet protocol

TN true negative

TP true positive

USN U.S. Navy

Acknowledgments

First, I would like to thank my wife, Audra, for her support and patience. Next, I would like to thank my advisor Dr. Barton, for his guidance and thoughts. Finally, I would like to thank Bruce Allen for his coding expertise and time.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

1.1 Motivation

"U.S. Hunts Chinese Malware That Could Disrupt American Military Operations."

"Ukraine Suffered More Data-Wiping Malware Last Year Than Anywhere, Ever."

"Lazarus hackers hijack Microsoft IIS servers to spread malware."

These are recent headlines of real world cyber threats that utilized malware in networks [1], [2], [3]. China, Russia, North Korea and other actors pose a serious threat to networks across the world. The U.S. military, and in particular, the Department of the Navy (DON), have been heavily targeted by these advanced persistent threats (APTs) [4].

In the October 2022 update of the National Security Strategy, U.S. President Biden lists "securing cyberspace" as a global priority. Cyberspace Operations are now so important to national security that the Department of Defense (DOD), and individual branches of service, release their own "Cyber Strategies" on a regular basis. In 2023, the DOD updated and released its classified *Cyber Strategy* [5], and later this year the U.S. Navy (USN) is expected to release its new *Cyber Strategy*. Moreover, in June of 2023 with NAVADMIN 143/23, the USN established its new Maritime Cyber Warfare Officer. Their goal is to develop their officer corps as skilled cyberspace operators with a vast understanding of cyber tools in order to defend vital U.S. interest and support allies [6].

This heightened threat in cyberspace, and the simultaneous advancement of artificial intelligence and machine learning, led to the main motivation behind the research in this thesis. The overarching idea is to leverage advancements in machine learning to detect malicious traffic in our networks.

1.2 Research Questions

1. How well can machine learning methods predict malicious and benign transmission control protocol (TCP) flows?

2. The traffic interaction graph (TIG) framework has been shown to capture 'burst' data within TCP flows. Can we harness this framework to accurately detect malicious flows that are hidden within an overwhelming amount of benign traffic?

3. Can we develop a graph neural network (GNN) model using TIG to classify TCP network flows with a high level of accuracy?

1.3 Scope

This thesis focuses on classifying network traffic flows as malicious or benign. We began with raw packet captures that were collected from two real-world networks. The first dataset ("Naval Postgraduate School (NPS) Enterprise Research Network (ERN)") consists of packets captured via the NPS ERN located at NPS, which is utilized by NPS students, faculty, and staff. The second dataset ("IoT-23") consists of packets captured by the Avast©AIC Laboratory, which were collected on various internet of things (IoT) devices. These devices were infected with different forms of malware and this dataset contains labeled packets (malicious or not). For the purposes of this thesis, the NPS ERN flows are assumed to be benign as they have passed through the NPS firewall in both directions. Additionally, this work focuses on the TIG framework as a way to describe TCP network traffic flows. Prior work in this realm has used machine learning (ML) methods on raw packet captures. This thesis, however, focuses on how the TIG framework improves (or does not improve) metrics on various ML algorithms. Specifically, we consider the random forest classifier (RFC), a simple convolutional neural network (CNN), and a simple graph convolutional network (GCN).

1.4 Contributions

This thesis contributes the following:

- We demonstrate that Traffic Interaction Graphs provide a mechanism for transforming TCP network traffic flows that result in highly accurate machine learning models.

- We demonstrate that GCN can model TCP network traffic flows after transformation to graph form.
- We demonstrate a novel method of model ratio splits where we inject various percentages of malicious traffic into real benign network traffic on a Naval network. We then test these various attack levels in a variety of machine learning models with exceptionally high accuracy.
- We provide a thorough literature review to future researchers who would like to understand and utilize graph neural networks.

1.5 Organization

This thesis is organized into chapters as follows:

Chapter 2 outlines the foundational knowledge needed to understand and use GNNs. We begin with a broad overview of machine learning. Then we cover all the essential definitions within graph theory in order to understand how graphs are used with machine learning. Finally, we provide an overview of the literature within the GNN field.

Chapter 3 describes our overall methodology. We begin with notes on the Python implementation of our models. We describe in detail the data transformation process where we convert raw packet capture data into TCP flows. We then provide an overview of the traffic interaction graph. Finally, we describe the three machine learning architectures that we use: the random forest classifier, a simple convolutional neural network, and a graph convolutional network.

Chapter 4 covers the experimental setup. We begin by discussing the two specific datasets that we use. We provide some analysis and visualization of these datasets. Finally, we provide results from each of the three models we ran.

Chapter 5 provides a summary of the entire thesis and provides several avenues for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: Prerequisites

2.1 Overview

There are two prerequisites to GNNs. One is machine learning (specifically, the Neural Network framework). The other is Graph Theory. We begin with the basics of Machine Learning and then discuss two popular methods: the Random Forest and the Convolutional Neural Network. Next we discuss in great detail Graph Theory by beginning with the definition of a graph and then discussing several other concepts that are commonly used within the GNN community. With the requisite knowledge in hand, we provide a broad overview of GNNs: first by describing some popular datasets, then by breaking down the various tasks that can be accomplished with GNNs. Finally, we provide a brief history of the field, current issues and cutting edge ideas.

2.2 Machine Learning

Machine learning is the art and science of training computers to help us use and understand data. There are three broad learning types of ML: supervised, unsupervised, and reinforcement learning [7], and these learning types can be used individually or together to model various problems. For this thesis, we will focus on Supervised Learning.

In Supervised Learning our data will need labels and our objective will be to predict the labels for future data. For example, consider a set of data that contains the following information about a large group of workers: age, gender, zip code, educational background, and income from the year 2021. If we are interested in predicting income based on the other variables, we can use income as our labels. Then we could use supervised learning algorithms to train a model to predict income based on the other information. The ML algorithms use this data and the respective labels to improve its predictions over time.

Only having access to this specific set of data, a model can learn the data a little too well and begin to *overfit*, thereby making it less accurate for new (unseen) data. Likewise, a model that is too simple can often *underfit*. The goal when developing a model then is to strike a balance between under and over fitting so that it performs best on future data.

2.2.1 Random Forest

A Forest is a collection of Trees. In Machine Learning, a Random Forest is a collection of Decision Trees which were built under specific random processes that we will discuss in a few paragraphs.

Decision Tree

A decision tree is a *tree*. Let us first describe how to use a decision tree once one has been created. Each node in the tree contains an equation based on a feature (e.g., $x_1 \leq 2.3$) or a final classification (e.g., malicious packet). For a given data point you begin at the root and based upon the rule presented at the root node you traverse down the tree in the direction based on the given data point and the rule. You continue this process until you reach a final classification node. This final classification is the prediction for that particular data point. See Figure 2.1.

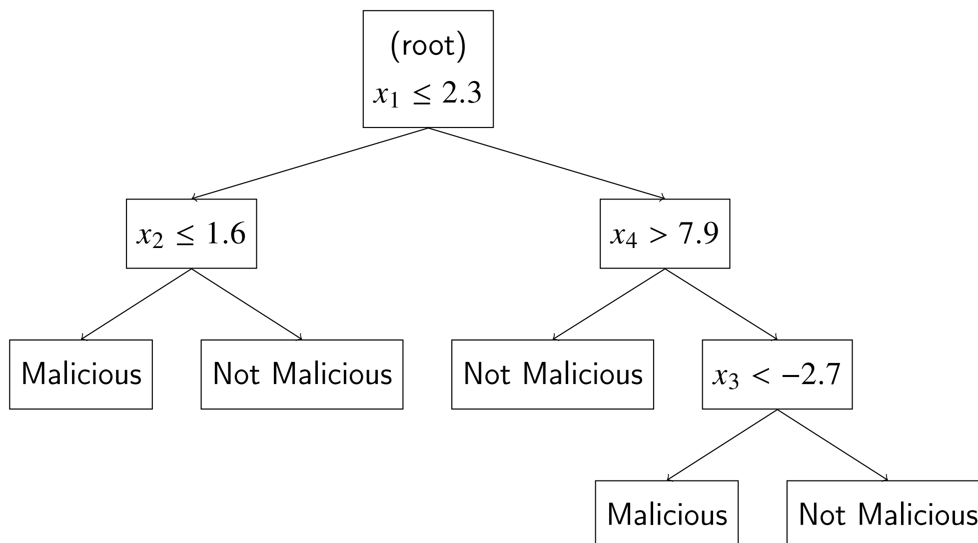

Figure 2.1. A decision tree.

The algorithm commonly associated with decision trees is Classification and Regression Trees or CART [7]. This algorithm works by splitting the dataset set into two distinct sets and then searching for an optimal feature and threshold pair (these are the 'decisions' we see in the nodes of the trees). The algorithm measures each feature and threshold pair by calculating the amount of correct and incorrect predictions. Once it has found the initial optimal threshold and feature, this process will repeat in the branch below (and continue to repeat) until told to stop running. There are several parameters that can be modified to prevent Decision Trees from overfitting data like the maximum depth of the tree or maximum number of leaf nodes.

Before we discuss the details of a Random Forest, let us describe *ensemble learning*. Ensemble learning refers to utilizing more than one prediction model and then aggregating the prediction in some way (e.g., by choosing the average of all the individual predictions or by choosing the most common prediction). Ensemble learning utilizes the concept behind the "wisdom of the crowd" [7]: that is, an aggregated answer from a group of non-experts is often more accurate than an answer by a single expert. This concept is heavily utilized with the Random Forest. In this case, the Random Forest will aggregate the predictions of numerous decision trees.

Now that we know that a Random Forest is just an ensemble learning technique utilizing several decision trees, we need to highlight another aspect that makes this classifier so powerful: randomness. The RFC takes advantage of random chance in two ways. First, the classifier will take several random samples of the dataset with replacement. This random sampling allows for the individual decision trees to be trained on a wide variety of the data in each individual instance (this sampling process is referred to as *bootstrapping*). Second, the RFC also uses random samples of the feature space to mix up the features that are used in each individual decision tree. Now that the training data has been sampled and the features have been randomly selected, the forest is constructed. The forest prediction will be an aggregate prediction which is based on the individual predictions made by each decision tree.

2.2.2 Convolution Neural Networks

CNNs differ from traditional Neural Networks in their inclusion and use of the *convolution* layer. This special layer acts as a filter, which is able to learn distinct features in the data. CNNs are often used in Computer Vision as these filters (commonly referred to as *kernels*) are able to learn distinct features within imagery (e.g., edges or faces). In addition to convolutional layers, CNNs utilize pooling layer(s) which provide two benefits: first, they reduce computational complexity of the model (e.g., memory usage and runtime) and second, these layers are able to abstract away the details learned in the convolutional layer and key-in on high-level features. CNNs have been shown to work well with data that is structured and grid-like [7], which is why we will use them with our highly structured network traffic flows.

2.2.3 Metrics

There will be several metrics used throughout this paper. Since we will be using binary classification models in this thesis, we will focus on metrics to measure binary outcomes. Additionally, we will define the following metrics in terms of what they mean when applied to the problem of network traffic analysis. The network traffic streams we will be making predictions on can be either **malicious** or non-malicious (benign).

A *true positive (TP)* occurs when the model predicted a malicious flow and this flow was indeed labelled as malicious.

A *true negative (TN)* occurs when the modeled predicted a non-malicious flow and this flow was labelled as non-malicious.

A *false positive (FP)* occurs when the model predicted a malicious flow but was wrong, and the flow was labelled as non-malicious.

A *false negative (FN)* occurs when the model predicted a non-malicious flow but was wrong, and the flow was labelled as malicious.

A *Confusion Matrix* is a quick way to organize each of the above four categories. In Table 2.1 we have the generic form of a confusion matrix and the usual positions of different classes. To illustrate how a confusion matrix can be helpful, let us assume that we have 1000 total network traffic flows, of which 10 are True Positives, 40 are False Positives, 20

are False Negatives and 930 are True Negatives. These numbers can be quickly summarized in the confusion matrix in Table 2.2.

Table 2.1. Confusion Matrix

		Actual Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Table 2.2. Confusion Matrix Illustration

		Actual Class	
		Malicious	Non-malicious
Predicted Class	Malicious	10	40
	Non-malicious	20	930

Now that we have these definitions, we can define some commonly used metrics.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy is simply the ratio of correct predictions to the total number of predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision is a measure of how your model performs at predicting false positives and is also known as the *false positive rate*.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall is also referred to as the *true positive rate* or *sensitivity*.

Precision vs. Recall: Note that unless the model is 100 percent accurate, there will be a trade-off between Precision and Recall. That is, an increase in one may result in a decrease

in the other (they cannot both simultaneously increase). Attempting to measure the balance between these two metrics led to the creation of the F_1 Score which is defined as follows:

$$F_1 \text{ score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Given our context of cyber threat detection, we have a balance to consider as well. False negatives are dangerous because they represent actual malicious traffic that was predicted as being non-malicious. False positives, however, waste time for cyber operators because they are analyzing traffic that was predicted to be malicious but in reality is not.

2.3 Graph Theory

2.3.1 What is a graph?

Graph Theory is the branch of Mathematics that deals with modeling relationships between objects [8]. In its most basic form, a graph is a set of nodes and a set of edges that connect some or all of those nodes in a meaningful way. For example, in a social network, a graph could model the relationship between a group of people, where each node represents a distinct person and an edge between two people represents that they are friends. Note in the example in Figure 2.2 that John is not friends with Jack (no edge connecting the two) and that Jenny is only friends with Jack.

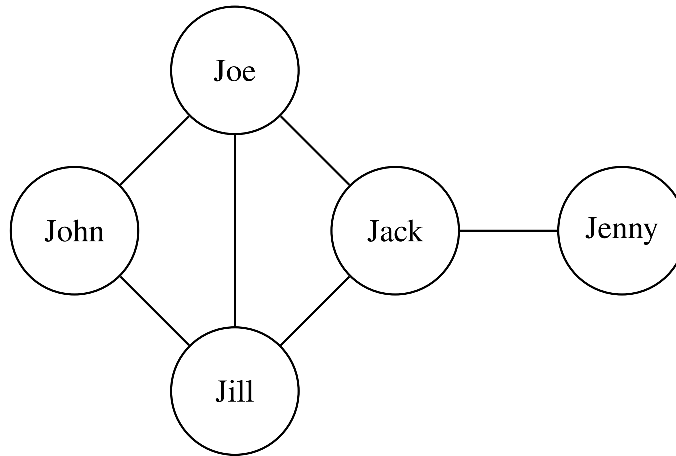

Figure 2.2. A graph modeling the friendships among a small group of people.

2.3.2 Graph definition and common terms

Several definitions are pertinent to the fields of Graph Theory and GNNs. These definitions are (mostly) standard throughout the field and as such will not be individually sourced. References for these definitions can be found in most introductory textbooks on Graph Theory. In particular, we used *Introductory Graph Theory* by Chartrand [8] and *Introduction to Graph Theory* by Wilson [9].

There are various ways to define a graph, but for the purposes of our work where we will be using **simple connected undirected** graphs with **no loops**, we define a graph as follows: a graph $G = (V, E)$ is an ordered pair where V is a set of distinct *nodes* (also referred to as *vertices*) and E is the set of *edges* (also referred to as *arcs*) connecting distinct pairs of nodes. Note that a *graph* is sometimes referred to as a *network*, but to avoid confusion we will not refer to one as such in this thesis.

As a concrete example, consider the Graph G defined as follows: $V = \{a, b, c, d\}$ and $E = \{ab, ac, ad, bc\}$. That graph can be expressed by Figure 2.3. We will refer to this graph frequently throughout this section.

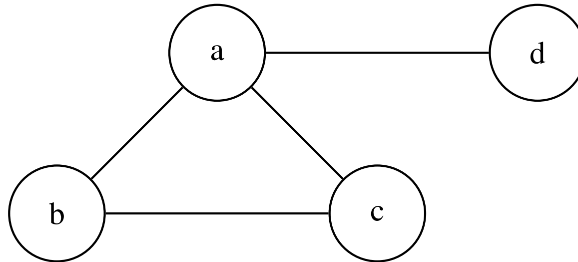

Figure 2.3. Graph G where $V = \{a, b, c, d\}$ and $E = \{ab, ac, ad, bc\}$.

In this example graph, the set of edges are simply defined as node pairs. Note that the order does not matter (edge ab is the same as edge ba for undirected graphs). Additionally, there are no loops allowed (a loop is an edge from a node to itself; in our notation that would be an edge aa).

Additionally, note that for a graph G, the set of nodes is frequently denoted as $V(G)$ and the set of edges as $E(G)$. The number of nodes or edges will be denoted $|V_G|$ and $|E_G|$, respectively (the subscript $_G$ may be dropped when it is clear what graph is being referenced).

Walks, Paths, and Connected Graphs

A *walk* between two nodes a and b is a sequence of nodes $(a, v_1, v_2, ..., b)$ beginning with node a and terminating with node b such that consecutive nodes are connected via edge(s). That is, there is an edge between a and v_1, an edge between v_1 and v_2 and so on. A *path* between two nodes is a walk that does not repeat any nodes. A graph is said to be *connected* if there is a path from each pair of nodes. Figure 2.4 highlights a path from node b to node d. Also, note that there is a path from each pair of nodes so this graph is connected.

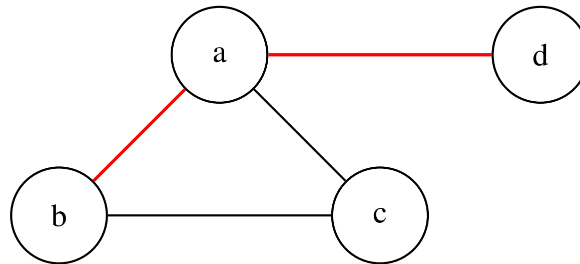

Figure 2.4. Graph G where $V = \{a, b, c, d\}$ and $E = \{ab, ac, ad, bc\}$.

Note that there are a variety of common and equivalent ways to describe the existence of an edge between two nodes a and b. We may say that *a is adjacent to b* or that *a is incident with b* or that there is an edge *joining* or *connecting* or *relating* a to b.

Directed and Undirected Graphs

Edges can either be *directed* or *undirected*. A directed edge starts at one node (the source) and stops at another node (the destination). Undirected (or bi-directional) edges are edges without a source and destination. If two nodes a, b are joined together by an edge then we can say that a is related to b and b is related to a. Another common way to think about bi-directional edges is that they are equivalent to having both directed edges. That is, a directed edge from a to b and a directed edge from b to a is equivalent to a single undirected edge between nodes a and b. A *directed graph* is a graph that contains only directed edges, and likewise, an *undirected graph* is a graph that contains only undirected edges.

In Figure 2.5 we have a directed graph. In edge ab, node a is the source and node b is the destination. Also note that between nodes b and c there is an edge for each direction. Additionally, there is no path to node a because a is not the destination node for any edge in $\overrightarrow{G}$.

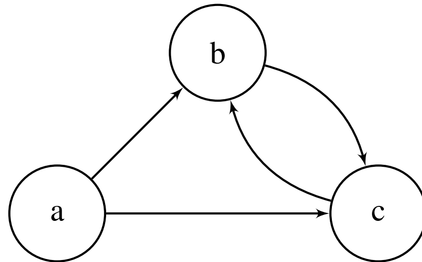

Figure 2.5. A directed graph $\overrightarrow{G}$.

Neighbors and degrees

For a node a of a Graph G, we define the neighbors of a, denoted $N(a)$ as the set of nodes that are adjacent to a. For the graph in Figure 2.6, $N(b) = \{a, c\}$ and $N(d) = \{a\}$.

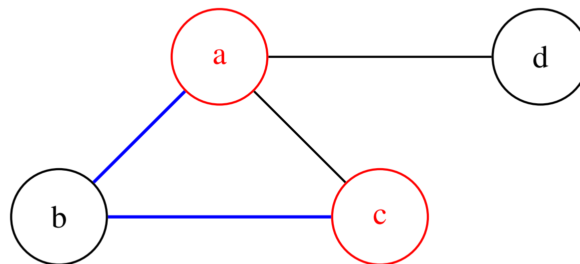

Figure 2.6. Graph G: Neighbors of node b are $\{a, c\}$ highlighted in red via edges in blue.

The degree of a node a is defined as the number of edges incident with that node. When thinking about a neighborhood, we can equivalently say that the degree of a node is the same as the size of its neighborhood (i.e., $\deg(a) = |N(a)|$).

In Figure 2.6 $\deg(a) = 3$, $\deg(b) = 2$, $\deg(c) = 2$, $\deg(d) = 1$.

In Figure 2.8, $\deg(a) = 2$, $\deg(b) = 0$, $\deg(c) = 1$, $\deg(d) = 1$.

Trees

Trees are a special type of graph common to Computer Science. A tree is a graph in which each pair of vertices is connected via a unique path. Another common way to define trees is that a tree is an acyclic (i.e., it contains no cycles) connected (there is at least one path between each pair of vertices) undirected graph. These definitions are mathematically equivalent, but in this thesis we will use the first. Trees are commonly drawn from the *root* down (as we saw in the decision trees section). The graph from Figure 2.2 is not a tree because it contains a cycle (connecting vertices $a, b, c,$). However, if we remove an edge (either $ab, ac, or bc$) as in Figure 2.7, then the result will now be a tree. A tree is a *binary tree* if each node has at most two branches. Said another way, a tree is a binary tree if for each node $a \in V$, $1 \leq deg(a) \leq 2$. If we remove edge ad from Figure 2.7 the result would be a binary tree.

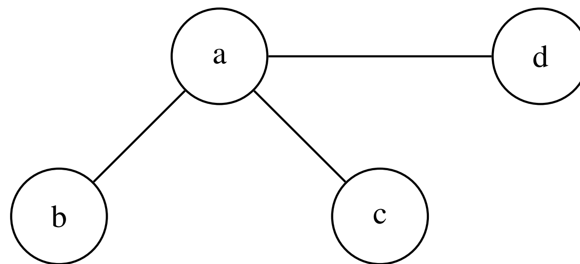

Figure 2.7. An example of a tree (after removing edge bc from previous graph examples).

Subgraphs

We can take a subset of the nodes and/or the edges to create a subgraph. Formally, a subgraph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$. For example, if we remove edge ab and bc but do not remove vertex b we have a disconnected subgraph of G. See Figure 2.8.

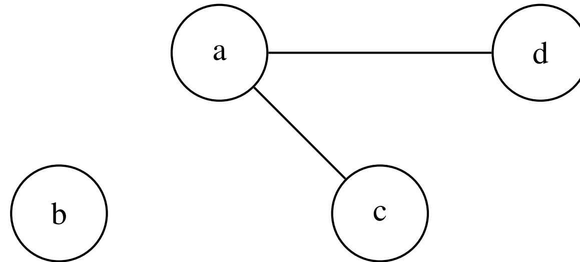

Figure 2.8. A subgraph G' of G which results in a disconnected graph (there is no path that can get to node b).

Graph Sameness - Isomorphism

It is helpful to know whether or not two graphs are the same, or if two subgraphs are the same. The notion used for this is called *isomorphism* within mathematics. In general, an *isomorphism* is a one-to-one correspondence from one object to another that preserves relationships. For graph theory, two graphs G and H are said to be *isomorphic* if there is a bijective function $f : V(G) \rightarrow V(H)$ that satisfies the following: for all nodes $u, v \in V(G)$, u and v are adjacent in G if and only if $f(u), f(v)$ are adjacent in H.

One practical concept that comes from this is that it does not matter how graphs are "drawn." They are still the same graph. Consider Figure 2.9 where we re-draw graph G on the left. Note that the edge from node a to node d crosses or overlaps the edge from b to c. This different drawing still represents the same graph G, which we see on the right.

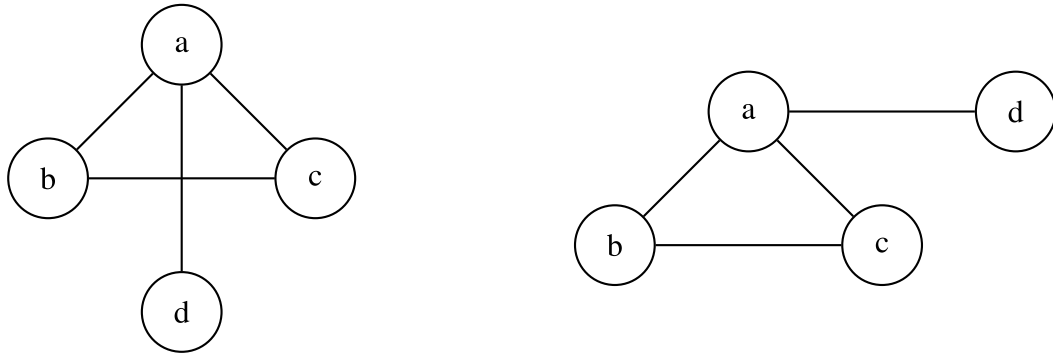

Figure 2.9. Two ways to draw the same graph.

2.3.3 Graph Representations

In this section we will discuss a few common ways to represent graphs numerically. These representations are frequently referenced in literature on GNNs.

Adjacency Matrix

The Adjacency Matrix of a graph is a useful way to describe the structure of a graph in a more rigid format. Recall that another way to describe a relationship between two nodes with an edge is to say that they (the nodes) are adjacent. Thus the adjacency matrix is a way to represent a graph through its edges (its adjacencies).

Let G be a simple undirected graph with a finite number of nodes (i.e., $|V| = n$). We define the *adjacency matrix* of G as an $n \times n$-matrix where each entry A_{ij} in the matrix is a 1 if there is an edge connecting node i and node j, and a 0 otherwise. In our definition of G, this matrix is necessarily square, symmetric (due to being undirected), and zero along the diagonal (due to no loops).

For our Graph G that we have highlighted throughout this section as an example, Figure 2.10 shows its adjacency matrix (assume that top to bottom and left to right goes from a to d in alphabetical order).

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.10. The Adjacency Matrix of Graph G.

Adjacency lists

One downside of the Adjacency Matrix is that for a large graph there will be a lot of empty entries within the matrix, making it sparse. One solution to this issue is to use adjacency lists. Then a graph G can be represented as a list of lists where for each node $a \in V(G)$, there is a corresponding list with all of the nodes adjacent to that node a (i.e., its neighborhood, $N(a)$).

For our graph G, its adjacency list is $[[b, c, d], [a, c], [a, b], [a]]$

Laplacian Matrix

Another useful way to represent simple graphs is with the Laplacian Matrix, which is defined as follows:

Again, let G be a simple undirected graph with no loops and a finite number of nodes n. Then we define each entry of the *Laplacian Matrix* of G as follows: each diagonal entry L_{ii} contains the degree of node i; each entry L_{ij} $(i \neq j)$ contains -1 if there is an edge connecting i and j, or is 0 if there is no such edge.

Stated another way, if A is the adjacency matrix and D is the degree matrix (where the diagonal contains the degree of each vertex), then the Laplacian matrix L can be represented as $L = D - A$. See Figure 2.11 for the Laplacian matrix of graph G.

$$
\begin{bmatrix}
3 & -1 & -1 & -1 \\
-1 & 2 & -1 & 0 \\
-1 & -1 & 2 & 0 \\
-1 & 0 & 0 & 1
\end{bmatrix}
$$

Figure 2.11. The Laplacian Matrix of Graph G.

2.4 Graph Neural Networks

This section will provide a broad overview of GNN literature. We hope that this section will help save time for future students at NPS who would like to learn about GNNs and apply them in their research.

2.4.1 Overview

In this section we begin by describing a few datasets that are heavily referenced in the GNN literature. We next discuss open graph benchmark (OGB), which is an outstanding resource with openly available, formatted, graph datasets, an application programming interface (API) to integrate with Python, and links to papers with GNN models and GitHub repositories with code.

Next, we discuss the idea of a *task*. GNNs differ from traditional supervised learning methods in that GNN models and their respective datasets are *task*-centric. That is, a GNN can make predictions at the **node**, **edge**, or **(sub)-graph** level. Similar to traditional unsupervised learning methods, we see techniques like anomaly detection and clustering except that these models are again broken down by their respective task (e.g., anomaly *node* detection). Similarly, datasets are built with the task in mind: for example, OGB splits up all of their datasets by task.

Next, we describe the early history of machine learning on graphs and how graphs were transformed in order to be used with existing ML models. Then we describe the first GNN model, a broader framework called "Graph Networks," and the Graph Convolutional Network.

We then address explainability. That is, attempting to explain why a model is making the decisions that it is making and what features or model-components it finds important. This is a burgeoning research area, and at the time of this writing, it is a difficult problem for GNNs.

Finally, we address heterogenous graphs. These types of graphs are able to encapsulate additional information by way of additional node or edge types. These models are extremely powerful and represent the cutting edge within the GNN community.

2.4.2 Common Datasets

Zachary's Karate Club

One of the most studied and simpler examples of a dataset modeled by graphs is Zachary's Karate Club. This graph consists of 34 nodes representing members of the Club and the 78 edges representing member interaction [10]. During the study of this social network, the club split into two factions and the author behind this study accurately predicted (for all but one member) which faction each member would join. The author did not use GNNs in any way, but this example is ubiquitous in the field [11].

CORA and CiteSeer datasets

CORA and CiteSeer are similar and popular datasets in the literature [12], and both are referred to as citation networks. They each consist of several thousand scientific research papers that have been broken down into relevant key-terms and categories. Each dataset also contains a dictionary of over one thousand words (words used within the scientific papers) and binary features for each word indicating its presence. Note that CiteSeer^{x} is also now a search engine (https://citeseerx.ist.psu.edu/) for academic papers (primarily in Computer and Information Sciences) with a variety of automated features designed to improve efficiency and assist researchers.

2.4.3 Open Graph Benchmark

Open Graph Benchmark is a website that hosts several large datasets from a variety of fields. These datasets have been processed and converted into graphs and can easily be loaded into Graph Neural Network algorithms via their API. Additionally, for each dataset, there is a

leaderboard with the top "scores" (e.g., accuracy) and links to the author's paper and GitHub repository. Moreover, these datasets are split up into each of the possible GNN tasks, which we will now discuss.

2.4.4 Tasks

Now that we have seen some common datasets that use graphs as models, let us now describe the various tasks that can be accomplished with GNNs. GNNs are different than traditional machine learning algorithms in that they are built around a specific *task*. We can use GNNs to make predictions about nodes, edges, or the graph as a whole. We can also look for anomalies within a graph or attempt to cluster nodes.

As we mentioned earlier, Machine Learning can be split up into *supervised* and *unsupervised* learning approaches. For Graph Neural Networks, *supervised* learning can be split up into the following three tasks: (1) Node, (2) Edge, and (3) Graph Prediction.

Node Prediction

Node prediction tasks occur where nodes are labeled, and the goal is to predict the label of a new (i.e., unlabeled) node.

For example, consider a large social media network where nodes represent users. One node prediction task could attempt to predict whether or not a user is a bot (and this would be based on its node features and connections to other nodes).

Edge Prediction

Edge prediction is sometimes also referred to as *link* prediction in the literature. Given a single graph, the task is to predict whether or not two nodes should be linked together by an edge.

For example, consider a basic social network where nodes represent people and edges represent connections among people (e.g., friendship, familial, co-worker). A social networking platform may find it useful to recommend connections to their users based on mutual connections. For example, Adam is friends with Beverly, Carl, and Donovan. Also, Emily is friends with Beverly, Carl, and Donovan. It then seems likely that Adam would also know

Emily as they share several mutual friends. That is, based on the mutual friends we would *predict* with a relatively high probability that there should be a link or edge between Adam and Emily.

Graph-level Prediction

For graph level prediction, the task is to make a prediction on the label of an entire graph (or sometimes subgraph). The input is often many individual graphs, and the task is given a graphs features make a prediction about it.

For example, consider the research we address in this thesis. We have millions of network traffic flows. Each individual flow is converted into a graph, specifically a TIG. The task is to determine the label for each individual graph: is the graph malicious or not (binary)?

Unsupervised learning tasks with GNNs revolve around detecting anomalies or clustering, which is similar to unsupervised learning methods found in traditional ML.

Anomaly Detection and Clustering using GNNs

Anomaly detection can be further divided into tasks like those seen before where we can attempt to find anomalous nodes or anomalous edges or anomalous graphs (or sub-graphs). Depending on the specific anomaly task, there are varying approaches. Most studied in the literature are methods to detect anomalous nodes.

Clustering tasks are the same as in traditional Machine Learning. For example, within a social network one could cluster similar users (i.e., nodes) based on some characteristic. Or, in an internet traffic example where each graph represents a network flow, one could cluster the flows into categories like social media, cloud computing, news.

2.4.5 Machine Learning on Graphs

We have now seen the various tasks that can be performed using Graph Neural Networks; now let us describe the challenges encountered with GNNs.

2.4.6 Graphs as Input

For traditional Machine Learning problems, the inputs are typically stored in a well-structured array (e.g., the MINST dataset has a rigid grid structure). However, with Graphs additional steps need to be taken in order to extract useful relational information in a way that can be used as input to Neural Networks. Early work in the field focused on methods that would convert a graph into a matrix or a set of arrays. One such idea was using the Adjacency Matrix that was discussed earlier. This method could work in some ML algorithms, but is limited in its ability to provide certain structural information about the graph (e.g., important nodes, or edges that serve as bridges). Further efforts in this realm involved extracting features via various statistical methods over the graphs (e.g., the Katz index which counts the number of paths between two nodes). These attempts were often limited or computational expensive which led to deeper methods known as graph representation learning (GRL) [13].

In GRL, one focus is on rich embeddings that can extract both structural information about a node and its local neighborhood. These embeddings can then be aggregated to provide embeddings for a sub-graph or graph. Additionally, these embeddings provide a more rigid array-like structure, which can then be used in Machine Learning algorithms.

2.4.7 The Graph Neural Network

Early attempts at using graphs as input to ML algorithms were typically focused on specific types of graphs, commonly directed acyclic graphs (DAGs). Additionally, these methods typically involved flattening the graph into a matrix representation which resulted in significant loss of the structural information about the graph. Scarseli et al. [14], in their seminal work, proposed the Graph Neural Network which was able to not only handle many different types of graphs (directed or undirected, cyclic or acyclic) but also used methods that preserved some of the underlying features of the graph structure.

Scarseli et al. [14] represents graphs in the usual way as objects (nodes) and relationships (edges between nodes), but they attach a *state* to each node which uses information from the neighbors of that node. This state is a vector that is based on the dimension of the neighborhood of nodes. The authors calculate the state by use of a parametric function utilizing the labels (information) from the node itself, its adjacent edges, and its neighbors,

and also states from its neighbors. This function is referred to as a *local transition function*, and the goal with this function is to determine how dependent an individual node is on its neighbors. Next, for a node, they use the calculated state and the node labels and apply another function to get the output for a particular node. This process is expanded out over all the nodes within a graph. The objective with all of these functions is to take a graph as input and return a real vector as output for each node. This mapping and the choice of functions are key for expressiveness and providing a way to use traditional ML ideas for learning. The authors provide specific learning functions and prove that these functions are differentiable and work with backpropagation (gradients converge).

In Battaglia et al., the GNN model is further generalized by removing the Neural aspect and using so-called "GN (Graph Network) Blocks" [15]. These blocks are designed to go from "graph-to-graph," meaning that they take a graph as input and produce a graph as output. Within blocks, various functions can be applied to capture certain properties from the graphs. The blocks can thus be stacked upon one another to form a network. The authors show how their GN block framework is more general than any other framework, and they provide the functions necessary to encompass popular networks like the Message Passing Neural Network and the Non-Local Neural Networks.

2.4.8 Graph Convolutional Networks

Graph Convolutional Networks are popular within the GNN community and are split into two main categories: Spatial and Spectral. Spectral approaches use Fourier transforms and attempt to filter the signal from the noise by means of a frequency response function. Spatial methods extract information from node neighbors via a node aggregation function, which updates the embedding of the graph [16]. For an in-depth framework that attempts to categorize and bridge the gap between spatial and spectral methods, see Chen, et al. [17].

2.4.9 Explainability of Graph Neural Networks

One difficulty in the GNN field is explainability; that is, being able to both interpret why the algorithm is making the decisions it makes and explain what the various layers are accomplishing and why they are needed. These concepts are important when considering the ethics of AI, when using AI on safety-critical systems, and to gain public trust [18]. There has been some ad-hoc work on attempting to explain basic GNNs such as Baldassarre

et al. [19], who uses a physical chemistry example involving solubility and molecular graphs and breaks down how to interpret actions conducted by a simple GNN model. Ying et. al. developed a tool, GNNEXPLAINER, to help explain GNNs [18]. This tool analyzes a GNN model and its predictions and then returns a subgraph and list of node features that are most important to the predictions. In Yuan et al., the authors develop a taxonomy for GNN explanation [20]. They split up explainability into two broad categories for GNNs: Instance-level and Model-level. *Instance-level* explanations use specific instances of input and output and then attempt to extract the most important nodes or edges or subgraphs. *Model-level* explanations attempt to explain the GNN model as an entity; that is, it attempts to explain what various layers are doing and how these layers are pertinent to the overall model. Model-level explanations have not been thoroughly studied. Overall, the field currently suffers from models that are difficult to explain and research that uses a trial-and-error style approach to finding what works [21].

2.4.10 Homogeneous vs. Heterogeneous graphs

One new direction the GNN field is heading towards is in the use of heterogeneous graphs. Within the GNN community, a *homogeneous* graph is a graph where the nodes and edges are of a single type. For example, in a social network the nodes often represent people, and the edges often represent connections between people. If we wanted to expand the information contained within this graph, we could, for example, add an additional type of edge: that is, we could have edges that represent friendship, marriage, siblings, etc. Using more than one type of node or edge takes the graph from being homogeneous to *heterogeneous*. This additional information allows for some powerful representations using graphs.

2.5 Network Traffic Analysis

Network traffic analysis is a broad field. In this thesis, we narrow the scope first by focusing on identifying malicious network traffic, second by only considering TCP traffic, and third by only using a few features (i.e., we use individual packet lengths, packet direction, and flow length, but it is common to see other features like port number or autonomous system number utilized).

2.5.1 Machine Learning Approaches

Because raw network traffic data is numeric, rigidly formatted, and abundant, ML methods are a natural choice when attempting to classify network traffic.

The RFC is a natural first choice when attempting to utilize machine learning methods to a field like network traffic analysis because it is an easy to implement algorithm. Additionally, the RFC performs well in a variety of contexts [7]. In Arif et al. [22], we see one of the first uses of a random forest classifier in network traffic analysis. They additionally use a variety of statistical techniques to classify various network attacks (e.g., denial of service attacks). The RFC performs extremely well compared to other complicated statistical techniques, and has now become a popular method for network traffic classification [23], [24], [25].

Additionally, CNNs are also commonly used for network traffic classification. Wang et. al [26] were one of the first to use CNNs to classify encrypted network traffic. They tested both one and two dimensional CNN frameworks on the publicly available ISCXIDS2012 dataset [27]. They found that one dimensional CNNs performed better than two dimensional on encrypted traffic flows [26] while also achieving state of the art results. Additionally, they made an argument that due to the sequential nature of network traffic flows, a one dimensional CNN is naturally more effective. Azab et al. [28] provide several other examples of CNNs used to classify network traffic. In Shahraki et al. [29], they took an ensemble approach with multiple CNNs. This CNN approach resulted in the authors correctly classifying internet traffic with nearly 98% accuracy on the Cambridge Internet traffic dataset.

2.5.2 Graph Neural Network Approaches

Graph neural networks and graph convolutional neural networks are relatively new approaches to tackling network traffic classification problems. Shen et al. [30] provides one of the first GCN approaches to classifying network traffic. They propose the TIG framework which we use in this thesis as a graph representation of TCP flows. Their work focuses on classifying TCP flows captured by decentralized applications. Their method provides highly accurate results for classifying encrypted traffic back to the source application from which it was sent.

Other researchers have also used various forms of GNNs to classify network traffic. Huoh et al. [31] leverage the graph structure to additionally capture meta and temporal features

of network traffic flows. These flows are routed through virtual private networks and are thus encrypted. Their GNN model outperforms common CNN and recurrent neural network (RNN) architectures, and achieved over 95% accuracy on their datasets vice 90% accuracy for traditional deep learning approaches [31]. Other GNN research in the network traffic realm has focused on classifying traffic found in mobile applications [32] or attempting to classify the *type* of network traffic (e.g., peer to peer vice streaming) [33].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

In this chapter we discuss the overall methodology behind this thesis. We first discuss the Python implementation decisions we made for the data visualization and the machine learning models we chose. Next, we discuss the data munging process to take raw packet capture data and transform it into network traffic flows. Then we describe the Traffic Interaction Graph framework, which we apply to our data sources and use as input to several machine learning models. Finally, we discuss the machine learning models that we use to make predictions.

3.1 Python Implementation

In any project that involves writing code, a choice needs to be made on the language. Python is used in several courses at NPS and is popular within the ML community. Additionally, Python has a large amount of documentation and online support, so we chose to use Python: specifically, Python 3.10.

Within ML, there are two popular frameworks: TensorFlow and PyTorch. In this thesis we will use both. We will use TensorFlow and sklearn for traditional Machine Learning methods: in particular, for an RFC and a CNN. We will use PyTorch for the GNN model. We chose TensorFlow for traditional ML models because of its ease of use and code base already established with prior work [34]. We use PyTorch over TensorFlow for the GNN model because PyTorch is better established and has more support and documentation available for GNNs than TensorFlow.

Some additional packages we use include sklearn for metrics, numpy and pandas for data loading and manipulation, and matplotlib for data visualization.

3.1.1 Graph Learning Packages and Libraries in Python

There are a handful of packages and libraries built using Python that are commonly used within the GNN community, summarized below:

- pytorch geometric (PyG): a package built on top of PyTorch for implementing common GNN models. PyG is one of the most popular packages for implementing GNNs.
- deep graph library (DGL): a library built in Python with options to build GNN models and data loading capabilities using either PyTorch or TensorFlow or any other deep learning framework within Python.
- TensorFlow_GNN: the official TensorFlow library for implementing GNN models using TensorFlow.
- NetworkX: a Python package designed around graphs (networks) as a data structure. This package does not have any built-in modules for GNN models, but it does allow the user to create or modify graphs, and it contains tools for the visualization of graphs. This package is analogous to pandas for ML, and each of the above GNN libraries can handle graph data from NetworkX.

We chose to use DGL in this thesis because it offers a flexible framework that can be built upon TensorFlow and PyTorch.

3.2 Data Transformation

Network traffic is commonly collected at the packet level. Using a tool like Wireshark or TShark to collect network traffic will result in a packet capture (PCAP) file. This file will contain a raw list of every packet collected over a network. Since our goal is to analyze and make predictions on network traffic flows, we first need to convert data from packets to flows.

3.2.1 Packets to Flows

First, let us define what we mean by a network traffic *flow* (also referred to as a *stream*). A flow can be thought of as a single conversation between two people: where words and sentences are sent back and forth and there is a clear start and end. In the network traffic sense, we can think of the speakers as a client and a host and the words and sentences as the packets going back and forth. Formally, we will use the 5-tuple definition from RFC 6146 to convert packets into flows [35].

5-tuple: (Source IP, Source Port, Destination IP, Destination Port, Transport protocol)

Thus, a flow is the sequence of packets identified by matching 5-tuple. The 5-tuple defined in this way will uniquely identify TCP flows. In this thesis we are interested in transmission control protocol/internet protocol (TCP/IP) flows and as such, after converting our datasets from raw packet captures to a collection of flows, we will filter out any flows that do not have a protocol of 6 (the protocol identifier for TCP). Note that this filter will not reduce our datasets in any significant way because a large portion of internet traffic is conducted over TCP. Additionally, because TCP is connection-oriented, it is natural to convert it into TCP flows [36].

3.2.2 Traffic Interaction Graph (TIG)

One crucial design element is the transformation of the data from network traffic flows, which are numeric numpy arrays, to a graph model format that we can use as input into a GNN.

Before we describe the details of a TIG, it will be helpful to understand and see a concrete example of the interaction between a client and a server at the packet level. Consider Figure 3.1 where the client initiates a TCP session to the server. The initial packets represent the TCP 3-way handshake [36], and the packets that follow represent the conversation between the two. The packet length shown (e.g.,-571) is the amount of bytes in an individual packet as seen on the wire (i.e., it includes header and payload). Regarding positive and negative signs, the authors of the TIG algorithm [30] use negative signs to represent packets sent by the client and positive signs to represent packets sent from the server (i.e., received by the client).

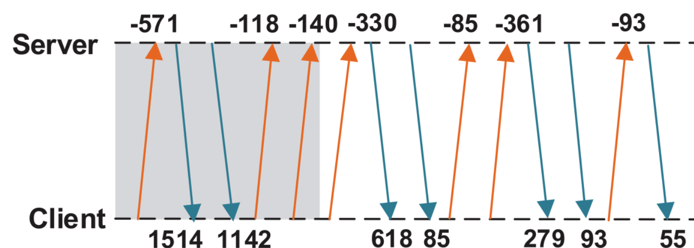

Figure 3.1. Packet-level Client-Server interaction. Source: [30, Figure 2(a)].

We will now describe the TIGs, which was developed by Shen et al [30]. In their paper, the authors not only propose and detail their algorithm for creating a TIG but they also achieve some significant machine learning results using network traffic data that has been converted to TIGs and then trained in a GNN model.

Shen et al., define their TIG as a three-tuple (V, E, L). Where V is the set of nodes, E the set of edges, and L is a set of non-zero integers. [30]

The set of nodes in a TIG represents the individual packets sent. The node label is the signed, necessarily non-zero, length of that packet. Again, packet direction is based on the sign (i.e., positive packets represent packets sent from the server to the client; negative packets represent packets sent from the client to the server). The signed packet length gets stored in the set L (labels).

Before we define the set of edges, it is necessary to first describe how the authors use the term *burst*. They use the word *burst* to describe the sequence of packets sent in a single direction before being interrupted by packets coming in from the other direction. For example, the first burst in Figure 3.1 is represented by the single orange arrow going from the client to the server with a packet length of -571. The next burst is the sequence of packets sent from the server to the client (the two blue arrows with packet lengths of 1514 and 1142). Then another burst occurs, this time containing a sequence of three packets from the client to the server. Note that a burst then can be as short as one packet. In this sense we can describe the bursts as a list of sequences containing packet lengths. For the example shown in Figure 3.1, this list begins: $[[-571], [1514, 1142], [-118, -140, -330], ...]$.

With this distinction in mind, the authors then define *inter*-burst and *intra*-burst edges as follows. *Inter*-burst edges connect vertices from one burst to the next: specifically, the first vertex in burst i is connected to the first vertex in burst $i + 1$ via an inter-burst edge and similarly, the last vertex in burst i is connected to the last vertex in burst $i + 1$ via an inter-burst edge. *Intra*-burst edges connect the vertices within a burst (in the order they occur). That is, if a burst contains three vertices a, b, c (in that order) then an intra-burst edge is formed between a and b and another intra-burst edge between b and c. Note also that if a burst contains a single vertex, then there are no intra-burst edges for that burst (there will be inter-burst edges connecting this burst to the burst before or after it).

Figure 3.2 shows the TIG corresponding to the flow from Figure 3.1. Note that the edges that connect vertices of a different color are inter-burst edges and the edges that connect vertices of the same color are intra-burst edges.

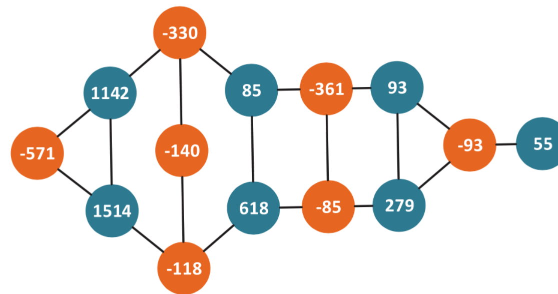

Figure 3.2. Traffic Interaction Graph for previous Client-Server interaction. Source: [30, Figure 2(b)].

The authors highlight four distinct strengths of their TIG. First, packet direction is known based off the sign of the node labels. Second, packet length is captured through the absolute value of the node labels. Third, packet order information is captured within the TIG. Fourth, packet burst information is revealed by the order and structure of the TIG.

The first three points are clear from Figure 3.2. Let us discuss burst patterns in more detail. The authors note that bursts "can vary significantly and thereby act as discriminative features" [30]. This idea is significant: if a machine learning algorithm is able to learn these burst patterns, then encrypted traffic sent over TCP could potentially be identified. That is, encrypted network traffic could begin to reveal patterns based on bursts (an idea referred to in the literature as *fingerprinting*). Thus, converting network traffic flows to TIGs has the potential to identify seemingly hidden patterns within malicious network traffic, and this identification is precisely our goal.

3.2.3 Final Data Product

One important note is that our final data is going to be in two different forms. First, for our GNN model we are going to use TIGs as described above and will use .dgl as the

data structure for storing these TIGs. Second, however, for traditional machine learning methods we cannot use a graph (e.g., TIG) as input, so we will attempt to capture as similar information as possible to the TIG but in a numpy (.npy) array format. Specifically, for our ML approaches we will use an array containing the signed packet lengths in the order they were sent. This approach will allow for knowledge of both packet direction and packet length. Moreover, packet order will also be captured by this format; however, we do not expect burst patterns to be captured as well as they would be with the TIG framework because there will be no indicator of the beginning and end of bursts. Additionally, since we will be dealing with strictly TCP flows, we will filter out any flows with fewer than the three packets required for the three-way handshake.

3.3 Machine Learning Models

We will be using three different models to make predictions on network traffic flows. All of the models will be making a binary prediction: malicious or not.

Note: for the RFC and CNN models, we will be using arrays stored via numpy as previously described. For the GNN model, we will be using TIGs stored as dgl files as described above.

3.3.1 Random Forest Classifier

We chose to use an RFC to establish a baseline for comparison. The RFC is an extremely common and easy to understand machine learning method and has been used in previous work on network traffic packet classification [34] and was also used by Shen et al. as a baseline comparison for their GNN model [30].

Our RFC will use default parameters from **sklearn.ensemble.RandomForestClassifier**. Of note: number of trees is set to 100, maximum number of features is set to the square root of the number of total features in the data (in our case this results in 5), and bootstrapping is set to true.

3.3.2 Convolutional Neural Network

We necessarily use a one-dimensional CNN, and we use a similar shallow architecture as seen in previous work with network traffic classification [34]. We include dropout layers to

fight over-fitting and pooling layers to extract high-level features. CNN architectures have been fruitful in pulling out patterns within data (e.g., images) [7], and our hope is that this CNN model will be able to discern some of the burst patterns in our network traffic flows. In Table 3.1 we list out the layers for our CNN architecture. Note that the input size is 25x1, and this represents the first 25 signed packet lengths of a flow. We will discuss why we consider just the first 25 packets in 4.1 Data.

Table 3.1. Convolutional Neural Network Model Architecture.

Layer	Parameters
Input	[25,1] vector
Conv1D	filters = 32
Dropout	30%
Conv1D	filters = 64
Dropout	30%
MaxPooling1D	pool-size = 2
Flatten	-
Dense	256 neurons, ReLU
Dropout	20%
Dense	1 neuron, Sigmoid

3.3.3 Graph Convolutional Network

We use a simple GCN architecture as both a proof-of-concept and as a baseline for future work. The task we will be addressing is *graph classification*. Recall that each individual network traffic flow has been converted into a TIG (i.e., a graph). Thus, our model will take as input a TIG and make a prediction as to whether or not that TIG is malicious or not (i.e., represents a malicious network traffic flow or not). Hence, as with our traditional machine learning methods, we are modeling binary classification.

The GCN architecture is as follows: We begin with a convolutional layer that takes in the node features of our TIG and outputs a 16-dimension hidden layer using ReLU as the activation function. Next is another convolutional layer that takes 16 features as input from

the previous hidden layer and two features as output (the number of classes, two for binary). This last layer will also aggregate all of the node data using the average (mean) of all the node features (i.e., the signed packet lengths). We use *Adam* as the optimizer with a learning rate of 0.01, *cross-entropy* as the loss function, and we train for five epochs. Most of these choices are standard or default.

CHAPTER 4: Experimental Setup

In this chapter we describe the experimental setup. First, we describe the data and the two sources of packet captures that we use. Next, we take an initial look at this data and gather summary statistics and visualizations. Finally, we describe and provide results from the machine learning models we discussed in the last chapter: specifically, a random forest classifier, convolutional neural network, and graph convolutional network.

4.1 Data

There are two datasets that we use in this thesis. First, we describe the NPS ERN data which was collected at NPS. Second, we describe the IoT-23 dataset which is a publicly available dataset containing packets sent from various forms of malware. Each dataset is initially a raw packet capture file (i.e.,.pcap) with typical network data such as timestamps, port numbers, IP addresses, TCP flags, and more. As we outlined in the previous chapter, we convert these raw packet captures to network traffic flows by using the 5-tuple definition and then convert these flows to either numpy arrays with the signed packet lengths of the first 25 packets as the features or we convert them into TIGs.

4.1.1 NPS ERN Dataset

The NPS ERN dataset is comprised of network traffic data that has traversed the NPS campus network which is utilized by authorized students, faculty, and staff. Figure 4.1 displays the general collection architecture we employ on this network. There are tap points on both sides of the firewall which allow us to determine which traffic was blocked and which traffic was allowed through the firewall. This data is initially collected via tshark and the packet capture consisting of several fields including IP address (source and destination), port number (source and destination), protocol (e.g., 6 for TCP), packet length (the size of the payload plus the header), and timestamps. Additionally, there have been several captures completed on this network, and we specifically use the 2209 captures which were collected from 18 hours of real web traffic on September 28th, 2022.

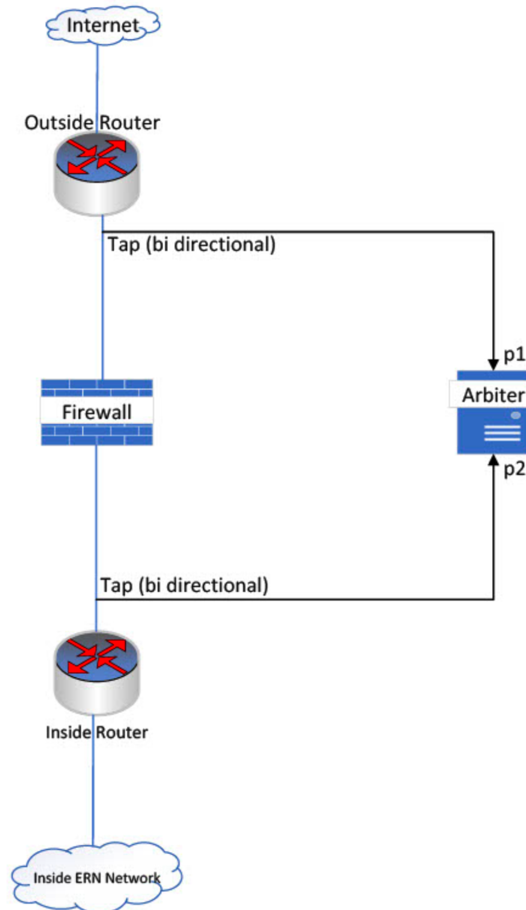

Figure 4.1. Collection Points and Firewall architecture of the NPS ERN. Source: [34, Figure 3.1].

There is one important distinction to note with how we use the NPS ERN data. Recall that we are interested in classifying TCP flows and that TCP flows begin with a three way handshake. Thus, in order to successfully complete this handshake, traffic must be allowed both in and out of the network. That is, this traffic must be allowed, in both directions, by the firewall. Therefore, we will assume that these flows represent benign traffic. We feel this is a reasonable assumption for two reasons: first, most network traffic is benign; second, the NPS ERN has strict firewall rules.

4.1.2 IoT-23 Dataset

The IoT-23 Dataset is publicly available online [37]. This dataset consists of malicious and benign network traffic that was captured on IoT devices [37]. In particular, they provide twenty distinct and labeled packet captures from three infected IoT devices. Moreover, these twenty captures were collected in 2018 and 2019 and are stored as pcap files. Note that the authors of this dataset also provide three datasets consisting of network traffic flows. The reason we chose not to use these flows is that they were processed via Zeek and our NPS ERN traffic flows were processed using a different method. Additionally, since the authors of the IoT-23 dataset provide the .pcap files, we are able to process their files in the exact same manner that we process the NPS ERN pcap files into network traffic flows. In terms of the labels used within this dataset, there are two types. First, a simple binary variable indicating whether the traffic was benign or not ('attack'). We use these 'attack' labeled packets to label flows as 'malicious' or not (i.e., a flow containing one or more 'attack' packets is labeled as 'malicious'). Of note, the authors also provide a more nuanced definition of the 'attack' label with several categories. These categories are based on the type of attack performed by the malware (e.g., Distributed Denial of Service or Botnet Command and Control). We will not be using these labels in this thesis, but this is one avenue for future work that we discuss in Chapter 5.

4.1.3 Flow Length

We have one final note to make about flow length. We ignore any packet data beyond 25 packets. This decision is based on the results found in Shen et. al. [30]. They experimented with various lengths and found 25 to be long enough to find burst patterns but not so long that it severely impacts training time and memory resources. Future work could look into expanding (or decreasing) flow length.

4.2 Data Exploration and Visualization

In this section we provide basic data exploration and visualization. We provide summary statistics and highlight relevant features of the data.

We have two datasets (NPS and IoT-23) and two data formats (numpy arrays and dgl graphs).

4.2.1 Numpy Data

The data stored as numpy arrays are used for the RFC and CNN. We summarize the data as follows.

There are 23,317,416 total flows for the NPS ERN portion, and there are 12,953,087 IoT-23 flows. Among the IoT-23 flows, 68.02% are labeled malicious (i.e., they contain at least one packet that was labeled malicious), and the remaining 31.98% are labeled benign.

Moreover, recall that we are using only the signed packet lengths as features. Additionally, we limit the number of packets considered in a flow to 25. We chose 25 to be consistent with the GNN model that we run, and due to the limitations and rationale there (see section 4.1.3). However, limiting to 25 here also helps reduce the memory storage requirements and speeds up processing time within our ML models. Future work could also look to expand this number.

In Figure 4.2 we have a histogram of all of the NPS ERN traffic flows and their respective flow lengths (i.e., the number of packets within a flow). Note that this distribution is bi-modal with flow-lengths of 3 and 25 being most common. Small flow lengths likely indicate failed TCP connections (e.g., the three-way handshake was not properly executed), and flow lengths of 25 are common because any flow longer than 25 is truncated down to 25.

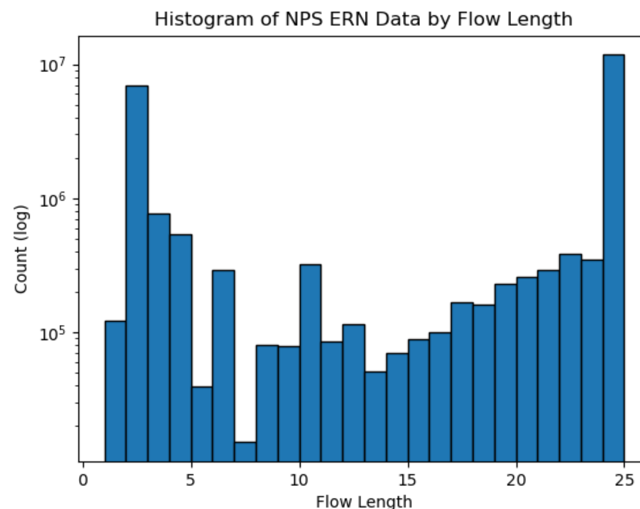

Figure 4.2. NPS ERN dataset: flow length vs. (log) count.

In Figure 4.3 we again see a bi-modal distribution at the two end points. There are some differences, however, when compared with the NPS flow lengths. First, the shortest flow length is four and there appears to be more traffic contained in the smaller flow lengths than in the NPS data. Second, there appears to be a declining trend: that is, as flow length increases, count also decreases (until running into the truncation value of 25).

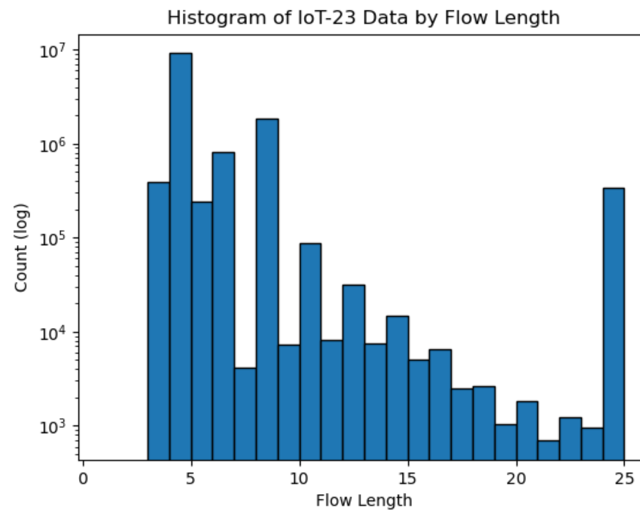

Figure 4.3. IoT-23 dataset: flow length vs. (log) count.

In Figure 4.4 we have side-by-side histograms comparing malicious and benign flows from the IoT-23 dataset. The malicious flow lengths exhibit an interesting trend where there is a spike at every odd number flow length. There is also a decreasing trend in both, but it is more pronounced in the malicious flows. Additionally, we see a spike at 25, which is the truncation value. Future work should consider the type of malicious attack (e.g., denial of service) and see if there are any trends within attack types.

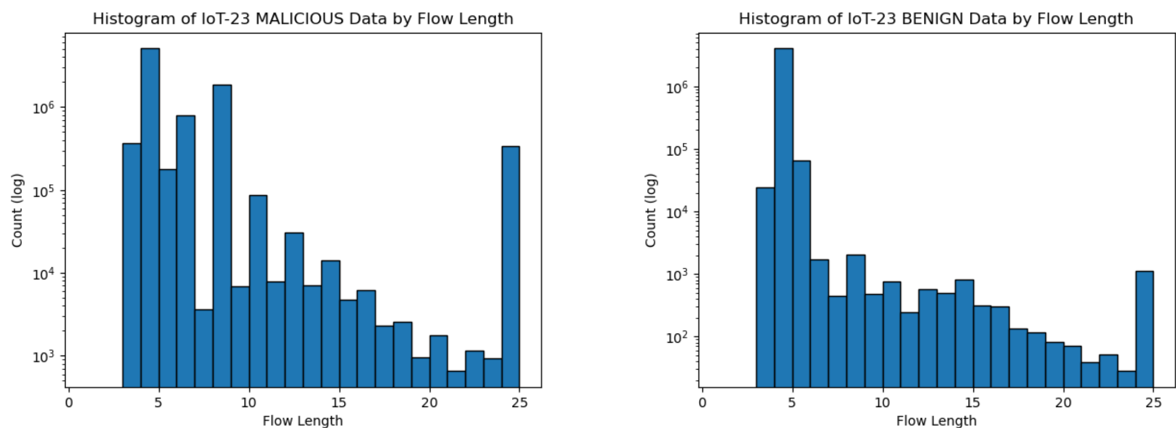

Figure 4.4. Flow length comparison of malicious (left) and benign (right) within the IoT-23 dataset.

Recall that the data we are entering into the traditional ML methods is an array of signed packet lengths (i.e., an array of integers). We analyzed and created histograms of the individual signed packet length across both malicious and benign flows on a column by column basis. Recall that each column corresponds to a specific packet: that is, each flow has 1 to 25 packets, in order. We looked at each distribution to see if there were any clear indicators, but none were found. In Figure 4.5 we see histograms for the signed packet lengths of the final packet (25). On the left are the malicious flows and on the right the benign flows, each from within the IoT-23 dataset. What we can see is that any packets with a 0 indicate that a flow has ended (that is, those flows contained fewer than 25 packets). Otherwise, there appear to be minimal distinguishing features between the two except that the malicious packet lengths include some higher values on the right tail.

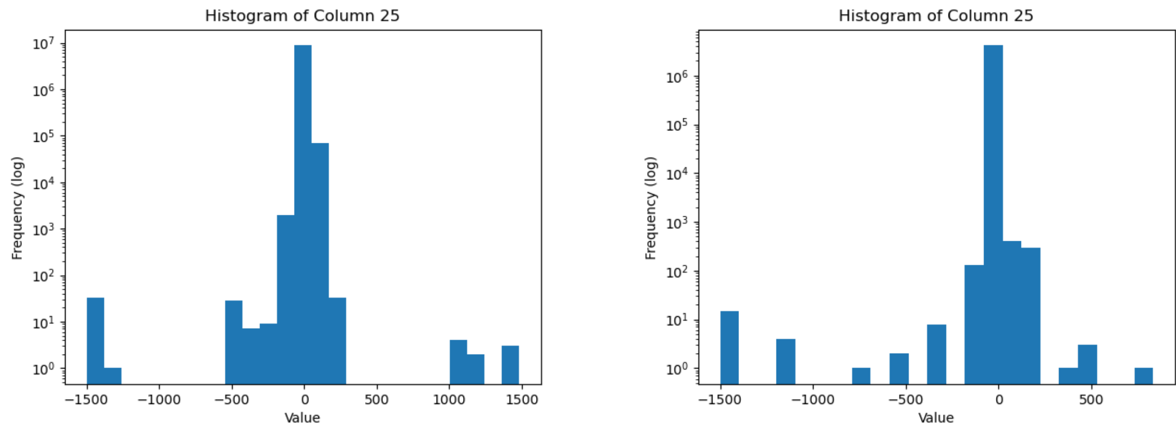

Figure 4.5. Signed Packet Length comparison of malicious (left) and benign (right) within the IoT-23 dataset for the 25th packet.

4.2.2 Graph Data

The graph data is built from the numpy data, so the distributions are identical and match what we saw in the previous section. There are some differences to highlight. First, the files are stored in the *.dgl* format. This format allows us to store signed packet length as a node feature. Additionally, we are able to store 'burst' data through the use of edges which we described in the traffic interaction graph section of Chapter three.

In Figure 4.6 we provide two actual TIGs from our datasets. Note that within a node there are two numbers. The top number is the signed packet length. The bottom number corresponds to the node within an individual batch and can be ignored for any data analysis purposes.

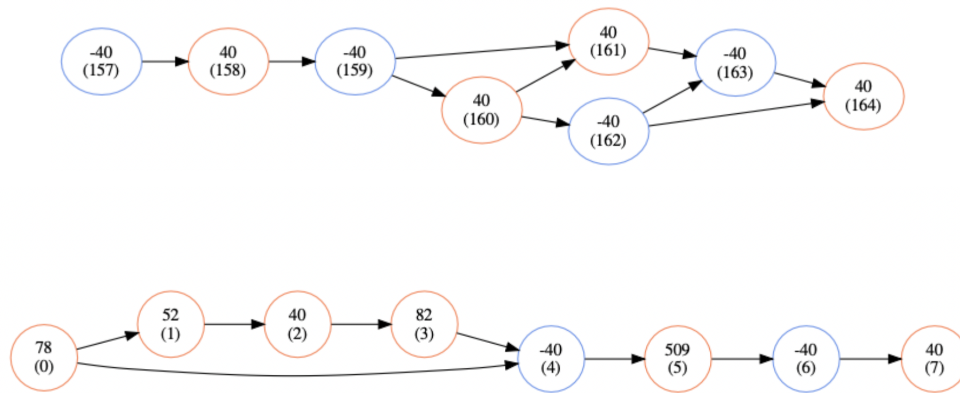

Figure 4.6. Two real life TIG examples from the IoT-23 dataset

4.3 Ratios

Our second research question address the idea of how well the TIG framework can be used (via burst pattern recognition) to identify malicious traffic among a heavy amount of benign traffic. Usually machine learning models perform better when there is class balance. That is, an equal proportion of each class (e.g., in our work this would look like 50% of the TCP flows being malicious and the other 50% being benign). While it is rare that data is perfectly split, it is generally considered better to be as balanced as possible [7]. When considering network traffic, the vast majority is benign. This creates an immediate imbalance. Rather than implement balancing techniques like random over sampling we want to evaluate the model on its ability to find the proverbial needle in the haystack. Thus, we take a different approach and run several iterations of machine learning models but with various ratios of NPS ERN data and IoT-23 data. In particular, we run each model using the following ratios:

- 99% NPS ERN to 1% IoT-23 (which we refer to as 99-1)
- 95% NPS ERN to 5% IoT-23 (99-5)
- 90% NPS ERN to 10% IoT-23 (90-10)
- 100% IoT-23 (IoT only)

Our rationale is that we would expect results to get worse the more imbalanced the overall dataset becomes. That is, we would expect results to be better for the 90-10 split than for

the 99-1 split (specifically in terms of precision and recall). Moreover, these various splits can be thought of as varying levels of attack. That is, a network under minimal attack could fall under the 99-1 split, whereas a network under much heavier attack might see a 90-10 split. Additionally, recall from above that the IoT-23 dataset is roughly 68% malicious, so a 99-1 split results in (on average) 0.68% malicious flows and 99.32% benign flows.

With respect to the IoT-23 dataset by itself, our goal is to provide results with this public dataset using the TIG framework, and then compare these results with other work that has been done modeling this dataset with machine learning.

Random Sampling

Because we have an incredible amount of data and a limited amount of resources (computing power, memory, and time) we are going to randomly sample from the two datasets before we input the data into our three models. In addition to keeping within computing parameters, this decision helped facilitate the splits of these datasets into the three ratio splits that we test over each model.

We use the Python **random** library, set the seed to 42, and use their **sample** method for all of our random sampling.

4.4 Results

In this section we go through each model and provide results for each split (e.g., 99-1, 99-5) with respect to the following metrics: accuracy, precision, recall, f1 score. Additionally, we also begin each section with a discussion of any design decisions that were specific to that particular model.

4.4.1 Random Forest Classifier

For the random forest model we needed to make a decision on how much data to use for training and how much to use for testing of the model. We chose to use a ratio of 70% training data to 30% test data which is a common split for such models when there is enough data to support. We did not use random sampling to split the data into each of the two categories but instead split over time stamp. That is, the training data came from the first 70% of timestamps (in chronological order) and the test set was chosen from the last 30%

of timestamps. The idea here is inherent to our overall goal: we want to use prior network traffic to make predictions on future network traffic. Additionally, by using this time-split we theoretically reduce the probability of identical traffic occurring in both the training and testing set (e.g., multiple flows to the same website within a short duration of time).

Note the NPS benign training set is always 16000000 and val always 6857142, and then we adjust the IoT-23 amount to fit for the corresponding split.

Table 4.1. RFC Split Metrics

Metric	99-1	95-5	90-10
Accuracy	0.998089	0.999056	0.998108
Precision	0.999520	0.999560	0.999547
Recall	0.981146	0.981344	0.981312
F1-Score	0.990248	0.990368	0.990346

IoT-23

The results below are for the IoT-23 data by itself. These flows are split in the same fashion as the previous split runs: that is, a 70/30 train/test split based on timestamp.

Table 4.2. RFC IoT-23 Metrics

Metric	Result
Accuracy	0.981414
Precision	0.999547
Recall	0.981651
F1-Score	0.990518

4.4.2 Convolutional Neural Network

In contrast to the RFC model, a CNN can utilize a train/validation/test split. This method requires a large amount of data but allows the researcher to have separate validation and testing portions of their data which increases the integrity and validity of the model as the

test data is used for final results but is completely new to the model. We chose a 60/20/20 split for our data, and again chose to split across timestamp. That is, the 60% training data came from the first 60% of flows (based on chronological timestamp), the validation portion of data came from the 60th through 80th percentile of timestamped flows, and the test data came from the final 20 percent of flows.

Table 4.3. CNN Split Metrics

CNN	99-1	95-5	90-10
Accuracy	0.999797	0.999564	0.999719
Precision	0.989144	0.997597	0.998792
Recall	0.990591	0.993675	0.998400
F1-Score	0.989672	0.995632	0.998596

IoT-23

For the IoT-23 only dataset, we again used a 60/20/20 training/validation/testing split. The results here were exceptionally high.

Table 4.4. CNN IoT-23 Metrics

Metric	Result
Accuracy	0.998264
Precision	0.999761
Recall	0.998491
F1-Score	0.999126

4.4.3 Graph Convolutional Neural Network

After running the above models as baselines we noted that the IoT-23 traffic set had a disproportionately high amount of malicious traffic in the validation set. That is, we believe the authors of that dataset must have executed more of the malware during the final twentieth

percentile of time. We believe this time-based split may have led to the models above performing better than expected on the validation set; future work with these datasets and models should verify this claim. This concern led us to use simple random sampling for the test-train split with the graph convolutional network instead of the time-based split used before with the RFC and CNN.

Additionally, we encountered memory constraints while running the GCN models with large amounts of data. This restriction led us to the following choices. First, we limited graph sizes so that they were between four and ten nodes. Most traffic was in between these two sizes already, and we can reasonably assume that valid TCP traffic should be at least three nodes in length due to the three-way handshake required. Next, we reduced the amount of data we used. We limited each of the splits to 2 million flows. We accomplished this limitation by first random sampling of the NPS traffic, only pulling 100 of the 500 total dgl files. Similarly, we randomly sampled the IoT dgl files. Next, as discussed above, we filtered out any TIGs with fewer than four or more than ten nodes. Finally, we randomly sampled from both the NPS TIGs and the IoT-23 TIGs in order to achieve the desired split (e.g., for the 99-1 split we sampled 1,900,000 from the NPS set and 100,000 from the IoT-23 graphs).

Table 4.5. GCN Split Metrics

GCN	99-1	95-5	90-10
Accuracy	0.982227	0.995603	0.963277
Precision	0.649176	0.605689	0.644024
Recall	0.998986	0.999012	0.999157
F1-Score	0.786959	0.754147	0.783211

IoT-23

For the IoT-23 only dataset, again we were constrained by memory. We were unable to run the GCN over the entire dataset, but we were able to use half of the entire dataset. That is, we ran the GCN on over six million IoT-23 TIGs. Like the split models above, we excluded any graphs that contained fewer than four nodes or more than ten nodes.

Table 4.6. GCN IoT-23 Metrics

Metric	Result
Accuracy	0.975638
Precision	0.999777
Recall	0.963362
F1-Score	0.981232

4.4.4 Model Comparisons

In this section we compare model performance side by side based on the data-split. Note that these comparisons are not direct. That is, there are key differences in the training datasets and splits. Moreover, one cannot claim that a single neural network is representative of all possible neural networks. We chose a simple model for both the CNN and GCN, but more complicated models could have produced better (or worse) results. Additionally, for the reader we highlight in bold the highest result across the various metrics.

Table 4.7. 99-1 Model Comparison

99-1	RFC	CNN	GCN
Accuracy	0.998089	**0.999797**	0.982227
Precision	**0.999520**	0.989144	0.649176
Recall	0.981146	0.990591	**0.998986**
F1-Score	**0.990248**	0.989672	0.786959

Table 4.8. 95-5 Model Comparison

95-5	RFC	CNN	GCN
Accuracy	0.999056	**0.999564**	0.995603
Precision	**0.999560**	0.997597	0.605689
Recall	0.981344	0.993675	**0.999012**
F1-Score	0.990368	**0.995632**	0.754147

Table 4.9. 90-10 Model Comparison

90-10	RFC	CNN	GCN
Accuracy	0.998108	**0.999719**	0.963277
Precision	**0.999547**	0.998792	0.644024
Recall	0.981312	0.998400	**0.999157**
F1-Score	0.990346	**0.998596**	0.783211

Across the board, the RFC and CNN perform exceptionally well. The GCN struggles with precision, but performs exceptionally well in accuracy and recall.

Table 4.10. IoT-23 Model Comparison

Model	RFC	CNN	GCN
Accuracy	0.981414	**0.998264**	0.975638
Precision	0.999547	0.999761	**0.999777**
Recall	0.981651	**0.998491**	0.963362
F1-Score	0.990518	**0.999126**	0.981232

For the IoT-23 dataset, all of the models perform exceptionally well, which is consistent with results seen from other researchers who have applied machine learning models to this dataset [38], [39], [40].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusion

In this chapter we provide a brief summary and discussion on what we have accomplished (including its relevance to the DOD), and we provide several avenues for future work.

5.1 Summary and Discussion

We began this thesis with a thorough review of the requisite knowledge required to understand graph neural networks. We then provided a broad overview of the entire GNN field, including common datasets, origins, various tasks handled by GNNs, and current issues. Our hope is that future students and researchers can take advantage of this section and apply GNNs to their research in useful ways.

Next, we described the TIG framework. Based on our results (and that of other researchers), we believe this to be a valuable way of describing TCP flows. We strongly encourage future researchers to utilize this framework, and we would like to see future work verify its utility through other avenues.

We additionally described in detail the two datasets that we used. It is highly valuable to include real internet traffic, and for the DOD, to use traffic going over military-affiliated networks like NPS. Data is the key to any machine learning endeavor. In the case of the IoT-23 dataset, we encountered a few limitations. First, the dataset was not well stratified across time. That is, based on chronological order, there were significantly more malicious flows found at later timestamps than at earlier timestamps. We believe this is due to the way the researchers collected the data for their own purposes (after running their final gauntlet of malicious traffic, they likely stopped capturing packets).

This data imbalance issue became a decision of trade-offs for us. Prior research in network traffic classification has split training and testing data along timestamps (using early timestamps as training in order to predict test traffic in future timestamps). However, our dataset is heavy on malicious traffic at the latter timestamps, which can result in fewer false positives when we run the final test data. Having fewer false positives will result in

an increase in precision. It is for this reason that I believe the GCN model performs not as well with respect to precision: for the GCN model, we randomly sampled from the entire dataset for the test/train split (as opposed to the RFC and CNN models where we split across timestamp). However, in defense of the traditional models and our methodology, we can think of our models having been trained on less malicious traffic, but then being tested on a highly malicious test set (e.g., simulating an adversary attacking our network), and performing exceptionally well. Future work should research other test/train splits for this dataset and see how the metrics (in particular, precision) change. Additionally, each model performs exceptionally well for the IoT-23 dataset, which is consistent with results from other researchers who have run models using this dataset. Our results were novel with regards to using the TIG framework and applying a GCN. With malware being present on more and more critical networks, highly accurate models can help the DOD quickly and accurately detect these cyber threats.

5.2 Future Work

Because most of the work conducted in this thesis was a proof of concept for using the TIG framework to classify TCP flows, ample work remains. There are two primary categories for building upon this work. First, improvement of the modeling. Second, improvement of the data. Outside of these two ideas, there is a potential software engineering project that applies the work we have done here in detecting malicious tcp flows into a real-time dashboard. Additionally, there is potential use for adversarial examples to bypass our detection methods. We discuss all of these ideas in detail below.

5.2.1 Improve the Modeling

The modeling we used in this thesis was simple: its primary purpose was providing baselines. Within machine learning, there are numerous ways to create more nuanced models. One can adjust the learning rate, number of epochs, or a handful of other parameters. Additionally, with the CNN and GCN model, there are numerous ways to add and utilize various layers to create more nuanced models. Additionally, future work could re-create the GCN model used in Shen et al [30]. Another option is to adjust the flow length. First, we could try to limit length to just the first 10 packets, and if the results are accurate enough, then this reduction provides a way to detect threats with fewer memory and time resources needed.

Second, expanding beyond 25 packets could provide more accurate burst detection (recall that a significant amount of flow lengths were 25 or greater). Finally, there is potential for multi-class modeling with the IoT-23 dataset. This dataset further broke malicious traffic into various categories: future work could attempt to detect threat type. As a warning with this last recommendation, the IoT-23 dataset contains nearly 20 categories, but only a handful of these categories contain enough examples to be of use.

5.2.2 Improve the Data

I believe this is the best route for future work.

Real-world data will always be best, but it is challenging to get such data with known malicious traffic on it, and in a manner that is controlled for evaluating model prediction. If such data cannot be found, the next best data source should be synthetic.

In a synthetic dataset, we could capture (or combine) real-world traffic with simulated malicious traffic. In particular, my recommendation would be to take a single piece of common malware and capture its network traffic under varying situations (e.g., a botnet command and control scenario). With this data in hand, we can then mix in real world data (like the NPS ERN dataset we used) and see how the models perform. This idea could be extended to other forms of malware, and in particular, ransomware. This idea additionally allows the future researcher to see if there are common burst patterns within these specific types of malware. Moreover, it may be possible to find subgraph structures that are common to malware by using GNNs. Also, since the data is synthetic, we could create enough of it to help simulate varying attack levels and also allow for more balanced training. Having more control of our data will allow additional confidence in the metrics and models created.

5.2.3 Real-Time Dashboard

For this work to be truly useful, it needs to be viable for real-time cyber threat detection. In its current state, the GCN approach is too cumbersome for any real-time utility. However, we do believe there is potential for the TIG framework with a simple random forest classifier to quickly and accurately detect malicious traffic. This project would entail writing software that would automatically convert raw packet to TCP flows and then transform those flows to TIG-based arrays and finally make a prediction from the already trained RFC. Additionally,

this project would be useful for network operators by providing a near real-time *dashboard* (think graphical user interface) with alerts and various network traffic statistics.

5.2.4 Adversarial Examples

We know that one weakness of using a hash function is that any change (even slight) to the software, results in a new hash. As these models get better and better at detecting malicious traffic, we must be concerned about the cat-and-mouse game that is played by network defenders and network attackers. In the realm of our thesis, I believe there is the potential for malicious actors to easily add or subtract packet length noise, or similarly, add or subtract unnecessary packets, as a way to evade detection by modern machine learning methods. With a synthetic dataset, a researcher could easily test this theory out by adding a random amount of nonsense to the payloads of malicious packets in order to adjust the packet length, or they could add additional packets with a random amount of payload in between their real malicious packets as another method.

List of References

[1] D. E. Sanger and J. E. Barnes, "U.S. Hunts Chinese Malware That Could Disrupt American Military Operations," New York Times, July 2023, [Online; accessed 02-August-2023]. Available: https://www.nytimes.com/2023/07/29/us/politics/china-malware-us-military-bases-taiwan.html

[2] A. Greenberg, "Ukraine Suffered More Data-Wiping Malware Last Year Than Anywhere, Ever," Wired, February 2023, [Online; accessed 02-August-2023]. Available: https://www.wired.com/story/ukraine-russia-wiper-malware/

[3] B. Toulas, "Lazarus hackers hijack Microsoft IIS servers to spread malware," BleepingComputer, July 2023, [Online; accessed 02-August-2023]. Available: https://www.bleepingcomputer.com/news/security/lazarus-hackers-hijack-microsoft-iis-servers-to-spread-malware/

[4] L. DarkOwl, "The Darknet Index: U.S. Government Edition," https://www.darkowl.com/wp-content/uploads/2022/02/DarkOwl-GovIndex_2018.pdf, 2018, [Online; accessed 02-August-2023].

[5] U.S. Department of Defense, "Fact Sheet: 2023 DoD Cyber Strategy," https://media.defense.gov/2023/May/26/2003231006/-1/-1/1/2023-DOD-CYBER-STRATEGY-FACT-SHEET.PDF, 2023, [Online; accessed 02-August-2023].

[6] U.S. Department of the Navy, "Navy Establishes the Maritime Cyber Warfare Officer (MCWO) Designator – 1880," https://www.navy.mil/Press-Office/News-Stories/Article/3441855/navy-establishes-the-maritime-cyber-warfare-officer-mcwo-designator-1880/, 2023, [Online; accessed 02-August-2023].

[7] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*, second edition ed. Sebastopol, CA: O'Reilly Media, Inc, 2019.

[8] G. Chartrand, *Introductory graph theory*, unabridged and corr ed. New York: Dover, 1985.

[9] R. J. Wilson, *Introduction to graph theory*, 4th ed. Harlow Munich: Prentice Hall, 2009.

[10] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. 452–473, dec 1977. Available: https://doi.org/10.1086%2Fjar.33.4.3629752

[11] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.

[12] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, “Collective classification in network data,” *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[13] W. L. Hamilton, “Graph representation learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159.

[14] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, “The Graph Neural Network Model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009. Available: http://ieeexplore.ieee.org/document/4700287/

[15] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, “Relational inductive biases, deep learning, and graph networks,” Oct. 2018, arXiv:1806.01261 [cs, stat]. Available: http://arxiv.org/abs/1806.01261

[16] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral Networks and Locally Connected Networks on Graphs,” May 2014, arXiv:1312.6203 [cs]. Available: http://arxiv.org/abs/1312.6203

[17] Z. Chen, F. Chen, L. Zhang, T. Ji, K. Fu, L. Zhao, F. Chen, L. Wu, C. Aggarwal, and C.-T. Lu, “Bridging the Gap between Spatial and Spectral Domains: A Survey on Graph Neural Networks,” July 2021, arXiv:2002.11867 [cs, stat]. Available: http://arxiv.org/abs/2002.11867

[18] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, “GNNExplainer: Generating Explanations for Graph Neural Networks,” Nov. 2019, arXiv:1903.03894 [cs, stat]. Available: http://arxiv.org/abs/1903.03894

[19] F. Baldassarre and H. Azizpour, “Explainability Techniques for Graph Convolutional Networks,” May 2019, arXiv:1905.13686 [cs, stat]. Available: http://arxiv.org/abs/1905.13686

[20] H. Yuan, H. Yu, S. Gui, and S. Ji, “Explainability in Graph Neural Networks: A Taxonomic Survey,” July 2022, arXiv:2012.15445 [cs]. Available: http://arxiv.org/abs/2012.15445

[21] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How Powerful are Graph Neural Networks?” Feb. 2019, arXiv:1810.00826 [cs, stat]. Available: http://arxiv.org/abs/1810.00826

[22] A. J. Malik, W. Shahzad, and F. A. Khan, "Network intrusion detection using hybrid binary pso and random forests algorithm," *Security and Communication Networks*, vol. 8, no. 16, pp. 2646–2660, 2012. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.508

[23] T. Wu, H. Fan, H. Zhu, C. You, H. Zhou, and X. Huang, "Intrusion detection system combined enhanced random forest with SMOTE algorithm," *EURASIP Journal on Advances in Signal Processing*, vol. 2022, no. 1, p. 39, May 2022. Available: https://doi.org/10.1186/s13634-022-00871-6

[24] N. Farnaaz and M. A. Jabbar, "Random Forest Modeling for Network Intrusion Detection System," *Procedia Computer Science*, vol. 89, pp. 213–217, 2016. Available: https://www.sciencedirect.com/science/article/pii/S1877050916311127

[25] T. T. Bhavani, M. K. Rao, and A. M. Reddy, "Network Intrusion Detection System Using Random Forest and Decision Tree Machine Learning Techniques," in *First International Conference on Sustainable Technologies for Computational Intelligence*, A. K. Luhach, J. A. Kosa, R. C. Poonia, X.-Z. Gao, and D. Singh, Eds. Singapore: Springer Singapore, 2020, pp. 637–643.

[26] 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.

[27] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, 2012. Available: https://www.sciencedirect.com/science/article/pii/S0167404811001672

[28] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digital Communications and Networks*, 2022. Available: https://www.sciencedirect.com/science/article/pii/S2352864822001845

[29] A. Shahraki, M. Abbasi, A. Taherkordi, and M. Kaosar, "Internet traffic classification using an ensemble of deep convolutional neural networks," in *Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility* (FlexNets '21). New York, NY, USA: Association for Computing Machinery, 2021, p. 38–43. Available: https://doi.org/10.1145/3472735.3473386

[30] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, “Accurate decentralized application identification via encrypted traffic analysis using graph neural networks,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2367–2380, 2021.

[31] T.-L. Huoh, Y. Luo, P. Li, and T. Zhang, “Flow-based encrypted network traffic classification with graph neural networks,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1224–1237, 2023.

[32] T.-D. Pham, T.-L. Ho, T. Truong-Huu, T.-D. Cao, and H.-L. Truong, “Mappgraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks,” in *Annual Computer Security Applications Conference*, 2021, pp. 1025–1038.

[33] T.-L. Huoh, Y. Luo, and T. Zhang, “Encrypted network traffic classification using a geometric learning model,” in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 376–383.

[34] M. Calnan, “Multi-Dimensional Profiling of Cyber Threats in Large-scale Networks,” Monterey, CA, Sep. 2022.

[35] M. Bangulo, UC3M, P. Matthews, Alcatel-Lucent, I. van Beijnum, IMDEA Networks, “Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers,” Internet Requests for Comments, RFC Editor, RFC 6146, April 2010. Available: http://www.rfc-editor.org/rfc/rfc6146.txt

[36] J. Postel, “Transmission Control Protocol,” Internet Requests for Comments, RFC Editor, RFC 793, September 1981. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[37] A. Parmisano, S. Garcia, and M. J. Erquiaga, “Aposemat IoT-23: A labeled dataset with malicious and benign IoT network traffic,” https://www.stratosphereips.org/datasets-iot23, 2020, [Online; accessed 27-July-2023].

[38] A. Singh, “Use of Machine Learning for Securing IoT,” Edmonton, Alberta, Canada, Dec. 2020.

[39] R. Gandhi, “Comparing Machine Learning and Deep Learning for IoT Botne,” San Marcos, CA, Mar. 2021.

[40] N. Stoian, “Machine Learning for anomaly detection in IoT networks : Malware analysis on the IoT-23 data set,” July 2020. Available: http://essay.utwente.nl/81979/

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE