DATA MINING AND DEEP LEARNING SYSTEMS FOR NETWORK TRAFFIC CLASSIFICATION AND CHARACTERIZATION AT SCALE

by

Maya Kapoor

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems, Software and Information Systems Track

Charlotte

2023

Approved by:

Dr. Siddharth Krishnan

Dr. Thomas Moyer

Dr. Ahmed Helmy

Dr. Xiang Zhang

ABSTRACT

MAYA KAPOOR. Data Mining and Deep Learning Systems for Network Traffic Classification and Characterization at Scale. (Under the direction of DR. SIDDHARTH KRISHNAN and DR. THOMAS MOYER)

You must have an abstract. The content of your abstract goes here.

You should compose the abstract using the conventions of your field. In many fields this section is typically 1-2 pages.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

If you decide to have an introduction page, your introduction text would go here.

Depending on the discipline or the requirements of the student's advisory committee, a preface may be included as a preliminary page.

CHAPTER 1: INTRODUCTION

Network traffic classification is critical in both network security and systems engineering. Cisco uses classification for enabling quality of service (QoS) features like fast-forwarding and buffering prioritization [4]. Network intrusion detection systems (NIDS) and prevention systems (NIPS) also rely on classification and identification techniques to locate malware signatures in payloads, perform dynamic access control [5], discover malicious flows and processes, and detect anomalies based on metadata features [6]. Classification is also used in internet traffic monitoring systems, or sniffers. Law enforcement agencies and intelligence organizations require the ability to collect, sessionize, and analyze streams of traffic. For law enforcement agencies, network packets contain information which can be routed back to crimes and cyber threats, and big data analytics can help reconstruct these traces into useful evidence [7].

1.1 Problem Statement

Network traffic classification covers a broad scope of problems; another limitation of existing research is that solutions tend to address only one or a subset of these issues without expanding to others. In this section, we introduce several of the most pressing problems to network traffic classification today and their relevance to real world systems.

1.1.1 Scalability

Machine learning and deep learning based systems are plagued by computational complexity and stream buffering requirements. In much of the existing research, entire traffic flows or certain portions of flows are required before classification can begin.

In a system at scale processing terabits of data per second, it is not practical to buffer this many streams in dynamic memory. Furthermore, classification speeds and models must be capable of accelerated computation through hardware or simple enough to be run in parallel quickly to keep up with line rate. For real-time classification, it may not be possible to wait for multiple packets in a single stream to identify a particular flow. Even in offline forensic analysis, entire streams may not be available or recoverable. Rather, the system must make a best effort guess without knowing the state of the traffic as to what application layer protocol is being carried for immediate parsing and processing. If possible, classifying the application layer traffic per packet would provide the lowest latency and highest throughput possible in the system.

1.1.2 Protocol Identification

Protocol and data exchanges in the modern internet can be complex over the lifetime of the session exchange, particularly for large amounts of transferable data. Specifically, routines like Voice-over-IP (VoIP) calls, file transfers and downloads, streaming services, and peer-to-peer sharing can transmit hundreds of thousands of packets of variable data over several minutes to hours. These exchanges are also not guaranteed to be one-to-one; for example, content servers

1.1.3 Tunneling and Routing

1.1.4 Traffic Profiling

1.1.5 Encryption and Compression

1.1.6 Port Obfuscation and Spoofing

1.1.7 Fingerprinting

CHAPTER 2: BACKGROUND

This section will introduce previous work related to deep packet inspection and the progression toward machine learning for network traffic classification. Related work toward specific techniques will be further discussed in the “Previous Work" subsections of subsequent chapters.

Not all research publications related to encrypted traffic classification can be fairly compared to one another; systems may classify the same data different ways. For example, DIDarknet [8] attempts to classify the same ISCX Tor/non-Tor dataset as Choorod et al [9], but DIDarknet profiles applications in the Tor traffic and Choorod et al's work focuses on darknet detection (only Tor or non-Tor). Many works use the ISCX VPN/non-VPN and Tor-non/Tor datasets or their original PCAPs and achieve higher accuracy and F1-scores than one another in various scenarios. These mechanisms require time-series based or flow features, which in real system application means capturing and analysis large portions of or entire flows in order to classify [10, 11, 12, 13, 14, 15, 16, 17, 18, ?]. Some systems have been able to cull this down to a few of the first packets of a connection [19, 20]. For a forensic analysis on a small dataset this may be sufficient; however, this does not scale to live capture systems. If the monitored network is sufficiently large enough (for example, on the order of terabytes of data per second), and traffic is expected to be classified and then processed by a follow-on application in real time, it may not be practical to buffer that amount of data and maintain the complexity of that many flows. In our threat model, we choose instead to prioritize making classification using only a single packet from any point in the overall traffic flow. We do not feed forward any information to the next classification or record any statistics about the packets over time or between arrivals. This is an implementation choice that focuses on performance optimization and achieving line rate as a single packet classification would lead to higher throughput as there is less strain on system resources and faster processing time without buffering. This trade-off may not always be the right choice depending on the real-world scenario. In some cases the priority may be accuracy and/or the system has resources capable of storing and processing the amount of data, or the amount of data many be contained. Having multiple systems capable of using different features is a more practical approach to the network traffic classification problem because it is unlikely one solution will work equally as well for multiple problems, and many real world applications may not afford all the features that one system requires over another.

2.1 Deep Packet Inspection

Deep packet inspection is the process of analyzing traffic data as it comes across the network. Packet headers contain information which can tell what type of application layer data is being transported in the packet. Furthermore, packet payload contents may be inspected for signatures matching protocols, applications, users, or other classes.

2.1.1 Port-based Identification

The Internet Assign Number Authority (IANA) requires that certain protocols be registered to certain ports [21]. In theory, this would allow traffic to be analyzed strictly at the transport layer to classify the packet; however, this technique has become increasingly unreliable due to nonstandard port usage and port translation. Additionally, some applications or users may choose to ignore protocol standards in implementations or design new applications which do not follow the standard. For example, a webmaster can easily host HTTP content on port 8080 rather than 80. Skype is known to use Web TCP to listen on port 80 although it is not HTTP traffic

[22]. Any peer-to-peer architecture also has the potential to operate on any range of ports, including ports intended for other applications [23].

Dynamic port numbers also pose a problem to port-based identification. Network Address Port Translation (NAPT) is a process that sits between the sender and the destination, allowing multiple computers on a network to use one global IP address. If a sender wants to listen over a specific port for a response from the internet and that port is not available within the NAPT set, the port will be translated to an unused number and mapped appropriately within the translation table. Therefore, when the destination is sending information back to the sender, the destination port will match what the NAPT set, not the port specified by the sender [24].

2.1.2 Payload-based Identification

Following noticeable decline in port-based identification accuracy, researchers began inspecting packet contents themselves for classification. Shallow packet inspection focuses on what can be derived from the physical, data link, network, and transport layers of packet encapsulation as described by the OSI model [25]. This then extended into deep packet inspection which studies the session, presentation, and application layers of a packet in order to derive the packet's content.

The increased usage of encryption protocols like TLS can render DPI ineffective [26], but it remains an efficient approach for plain text traffic. Shallow DPI can also be used to analyze public headers in packets. Many applications will send control, setup, and session management messages in the clear, which may provide contextual information or protocol type indication [27, 28, 29]. Additionally, public Wi-Fi access points may also pass unencrypted traffic which sniffers can then decode and analyze [30]. The expansion of Internet of Things (IoT) devices has caused rapid growth in new protocols and formats which currently outpaces the security regulations for them. Many of these devices like Google Home and Amazon Echo do not set encryption standards, have privacy policies in place, or are yet regulated by cybersecurity laws and will send and store data messages in plain text [?, 31, 32]. Older systems such as those found in factory SCADA and PLC networks may also not support encryption [33] and thus may be inspected.

Research has shown that for some applications, encrypted payloads still hold common data sequences which can be used for signatures. For example, Bernaille et al [34] found that a specific 15-byte sequence always occurred at the beginning of a Skype login server (LS) message, which could be used as a signature for application detection despite Skype being end-to-end encrypted even in its signaling messages. The HeaderHunter system [35] utilizes DPI pattern matching on packet metadata found in the headers of encrypted traffic, specifically packet size, direction, and TCP information. This and other systems [36, 37] require a full session capture in order to classify packets after the fact, so are not applicable to single-packet classification. These examples and their shortcomings further assert the relevance and potential of our system and payload-based identification as a whole.

2.1.3 Flow-based Identification

Developers have also examined using other heuristics from protocol flows to derive knowledge about packets and their protocols. These statistics-based identification methods typically focus on mean, minimum, and maximum of packet sizes, the number of packets seen in a given exchange, the time between packets, burst rates, IP/port numbers, TCP flags, flow duration, and so forth [36, 26, 38]. In the existing literature, most use machine-learning or statistics techniques that study time and flow-based features of packet streams [39, 40, 41, 42, 43, ?, 18]; however, in real-time network engineers may not be able to wait for or buffer entire packet streams before they can classify packets and send them to follow-on processing. In forensics, analysts may not have access to full streams. Many machine learning methods are not applicable for stateless packet inspection as they require time-based or flow-based features. Thus, per-packet solutions for both encrypted and non-encrypted data are a pressing research need regardless of flow-based solutions.

2.2 Network Traffic Classification

A machine-learning based approached can be used to train models to make informed decisions based on extracted features like packet statistics or common substrings [?]. For example, video streams can be classified using a NaÃ¯ve Bayes approach and extracting features related to Quality of Service requirements and adaptive streaming [5]. SVM has also been widely studied for this purpose [42, 40], but has significant drawbacks such as high computational complexity, high-retraining time, and reduced performance compared to other models like K-Nearest Neighbors [39]. Decision tree classification methods have also been used to classify traffic with high performance on large data sets; C4.5 is one of the most utilized methods used in this domain [44, ?]. In recent research, neural networks have also been adapted to solve the classification problem with greater success when compared to C4.5 [?]. Word embeddings have also become a practical approach even in the stateless context with Packet2Vec [45].

CHAPTER 3: REXACTOR - REGULAR EXPRESSION SIGNATURE GENERATION FOR STATELESS PACKET INSPECTION

3.1 Introduction

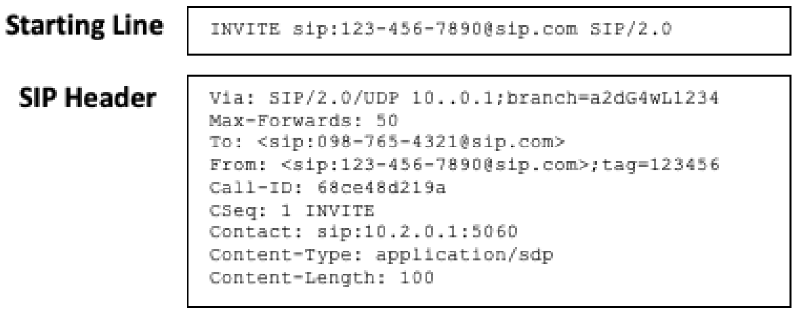
Regular expressions, often referred to as regexes, are sequences of characters used to specify search patterns in text for string-based searching algorithms. Stephen Cole Kleene first developed regular expressions as a derivative of study in automata theory and formal languages in theoretical computer science. Today, regular expressions are powerful notations used in many tools including UNIX programs like grep, awk, sed, and vi; search engines like Google, Yahoo, and Bing [], and deep packet inspection software like Snort [], RE2 [], and TRE []. Universal standards exist for defining regular expressions, such as Perl [], PCRE [], and POSIX [] syntaxes.

Regexes specify a subset of possible strings derived from the alphabet based on conditions described by the syntax of the expression. Operators exist which define the patterns. For example, a vertical bar (gray|grey) would indicate alternatives and match the string literals “gray" or “grey." The regular expression gr(a|e)y would be considered an equivalent regular expression and also match both these literals. Quantifiers specify how many occurrences of a character will appear in the given string. The question mark indicates zero or one occurrences of the preceding character. For example, (too?) would match “to" or “too." The Kleene star specifies zero or more occurrences of the preceding character; the regex a$*$ would thus match a string containing zero up to an infinite number of as, versus the Kleene plus regex a$+$ which would match strings containing one up to an infinite number of as. A specific number of occurrences or a range may also be specified with curly brackets. These are just a few examples of regular expression operators, not including POSIX groups or other such syntax-specific patterns.

Regular expression pattern matching is a text-based search solution which works best with data which is strongly signatured. Specifically, the method names, keywords, common return codes, and string tokens that frequently occur in packets make it possible for network engineers to come up with regexes that can match specific protocols for application layer identification. One example of strongly signatured packet data which can be matched with regular expressions is HTTP headers. The regular expression, (GET|HEAD|POST). $*$ HTTP/1.(1|0)x0Dx0A, will match all HTTP packets containing a method signature like “GET/ index.html HTTP/1.0", or “HEAD /index.html HTTP/1.1."

Machine learning research has found that for protocols which have commonalities across messages, unique features tend to be found in the first N bytes of packet payload [46]. In many mainstream application protocols such as POP3, SMTP, HTTP, SIP, XMPP, IRC, and others, the first line of payload data contains methods, commands, version numbers, protocol names, and other potential features. Figure 3.1 shows this starting line in the SIP protocol. Various protocols call this feature by different names; for example, HTTP calls this line “request-line" or “response-line" [47]. FTP refers to this line as “request" or “response" command [48]. For consistency, we refer to this portion across protocols as the starting line. In testing other protocols which do not necessarily follow a request-response paradigm, we adapted the framework to include other message type options. For example, XMPP utilizes three kinds of stanzas which present unique features: IQ, message, and presence [49].

Creating regular expressions for content filtering or pattern matching is a complex problem that often requires subject matter expertise and manual intervention. Cross-examining multiple packet payloads and examining technical reference documents like RFCs is a time-consuming but necessary process to derive accurate and effective regular expressions. Furthermore, these signatures must be regularly maintained,


Starting Line
SIP Header

```
INVITE sip:123-456-7890@sip.com SIP/2.0
```

```
Via: SIP/2.0/UDP 10..0.1;branch=a2dG4wL1234
Max-Forwards: 50
To: <sip:098-765-4321@sip.com>
From: <sip:123-456-7890@sip.com>;tag=123456
Call-ID: 68ce48d219a
CSeq: 1 INVITE
Contact: sip:10.2.0.1:5060
Content-Type: application/sdp
Content-Length: 100
```

Figure 3.1: Example SIP packet.

tested, and updated. Regular expressions become immediately ineffective for example if version numbers are added or changed. In the cybersecurity domain, malware signatures may be easily subverted through simple code or string manipulation so permutations must be considered, as well. In order to expedite this process and increase signature scope, researchers have leveraged knowledge discovery techniques and applied data mining to extract common features from packet payloads and header contents as input into regular expression signature generation algorithms.

3.2 Previous Work

Applications for regular expression signature matching to packet payloads originate in content inspection for malware detection. Early work in automatic signature generation for polymorphic worms such as Autograph [50] provided a foundation for signature-based analysis. This expanded into detecting personally identifiable information as well as policy violations such as copyright infringement, inappropriate or sensitive information on enterprise networks, and censorship []. Additionally, Tang et al [51] found multiple sequence alignment to be effective in rewarding consecutive substring extractions and tolerating noise in traffic. SigBox [?] introduced the technique of generating substring tokens from packet payloads and applying Apriori data mining to find the most frequent ones. The related CSP Algorithm [52] and works by Wang et al [53], Szabo et al [54], LASER [55], and AutoSig [56] all include feature extraction and sequence alignment. Wang et al describe their system as a four-stage process; we generalize this concept to all these solutions and our own in order to compare various approaches.

In the first step, data is pre-processed in order to prepare it for use at the next stage. This typically involves extraction of certain fields or N number of bytes from the packet payload. For all these systems, sessionization of packet flows and sometimes defragmentation and reassembly based on TCP/IP header values is required [?, 53, 52, 54, 56, 55]. Once the data is collected and prepared, the second stage finds common substrings across packet data and/or protocol flows. The systems examined which perform feature extraction use either a subtree approach [53], a longest common substring algorithm [56, 55], motifs [54], or sequential pattern mining [?, 52]. Third, a method of alignment is used to align data based on commonalities between packets. Wang et al, Szabo et al, Vinoth et al [57] and LASER use bioinformatics approaches to perform these alignments. A substring tree can also be used for this purpose [56]. Some of these alignment strategies are additionally informed by a scoring matrix influenced by the tokens derived in the previous step [53, 54, 55]. Once the sequences are aligned in an optimal manner, in the fourth and final stage the systems convert their results into regular expressions.

3.3 Methodology

We use a modified version of Apriori algorithm combined with frequency position tables in order to calculate single byte-length tokens and frequently occurring substrings using choice operators and position constraints [53]. RExACtor also uses genetic sequence alignment to find commonalities in payloads and encodes this alignment in order to create a regular expression from it. Our system performs better than previous work in memory space allocation by additionally using position frequencies to specify character classes and length constraints instead of generic wildcards. This creates real-time performance optimization in the scanner. Furthermore, unlike previous solutions our system requires no packet buffering or traffic flow information to do its identification.

RExACtor is made up of component modules which perform various stages of data processing and analysis in order to create optimal regular expressions. The flow of the system is shown in Figure 3.2. RExACtor takes in as input packet capture (PCAP) files and extracts session payload strings for supported protocols. The input can be of any mix of traffic types, and is therefore well-suited to learning in the wild from real network captures. This extraction layer takes in parameters from the command line interface to select a supported protocol and message type to specifically isolate training data from the mixed input. It also offers an extraction option for the full data layer for analysis. For our experiments, we selected HTTP, SIP, and RTSP as protocols and extracted request and response session payloads for each of them.

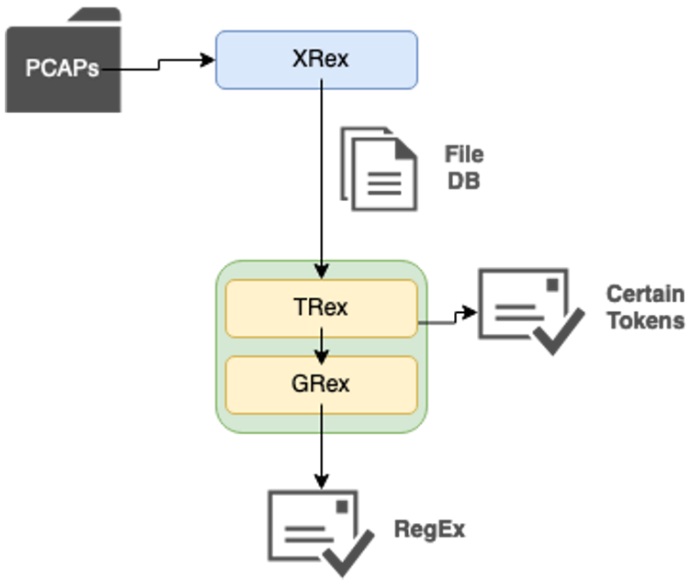


Figure 3.2: High-level diagram of signature construction in RExACtor. The system produces certain tokens and regular expressions which can be used with the HRex Scanner or any external filtering/sniffing application.

3.3.1 TRex: Apriori Tokenization of Packet Payloads

Inside the starting lines, some substrings appear frequently across packets and thus can be used as string literals in signatures. In the SIP protocol, the line “SIP/2.0" appears at the end of all starting lines in all our training data SIP requests. In HTTP, the protocol name and version “HTTP/1.1" appear consistently, as well. We use a modified version of the Apriori Algorithm [58] in order to find frequently occurring substrings, or *tokens*, in our packet strings. Shim et al also use Apriori in SigBox for content strings [**?**].

Apriori is an algorithm used for finding frequent item sets and association rules from databases. The algorithm uses a bottom-up approach to combine items into incrementally larger item sets, keeping only those sets which meet some minimum threshold of support. The support of an itemset in the database is defined as the number of instances of the itemset in the transactions over the total number of transactions [58].

$$\text{Support(X)} = \frac{\text{Instances of X in transactions}}{\text{Number of transactions}}$$

In the formal definition provided by Rakesh Agrawal [59], let $I = \{i_1, i_2, i_3...i_n\}$ be a set of items of length n and $D = \{t_1, t_2, t_3...t_n\}$ be the set of transactions known as a database. Every transaction t_i has a unique identifier in D and $\forall t_i : x \in t_i \to x \in I$, or every transaction is a subset of items in I. Apriori relies on the anti-monotonicity of the support of itemsets which assumes that all subsets of a frequent itemset must also be frequent, and all supersets of an infrequent itemset must also be infrequent.

We mine tokens using a modified Apriori approach. Each packet string derived from pre-processing is treated as an individual line in the database. TRex uses a sliding

Algorithm 1: Modified Apriori Algorithm [59]

```
Result: Frequent Itemsets
$k = 1$, $I_k = \{$frequent item sets of size 1$\}$;
while $(I_k \neq \emptyset)$ do
    $C_{k+1} = apriori\_gen(I_k)$;
    for $t \in D$ do
        $C_t = subset(C_{k+1}, t)$;
        for $c \in C_t$ do
            c.count++;
        end
        $I_{k+1} = c \in C_{k+1} \mid c.count \geq minsup$;
    end
    $k = k + 1$;
end
tokens = $\cup_k I_k$;
return prune_substrings(tokens);
```

window algorithm of length k to create substring items which are grouped into a list treated as a single transaction. We initialize the window size to $k = 2$ in order to prevent likely frequent but meaningless single-byte tokens and set the maximum itemset length $l = 1$. The first iteration calculates the frequency of substrings of length $k = 2$ and each subsequent loop increments k until no more frequent itemsets (substrings) are found.

Because the order of substrings matters in regular expression matching, we limit the maximum itemset length to 1. We increase item size instead by increasing the window size, thereby preserving order of characters in the modified algorithm. At each iteration, we also prune the resulting token set by replacing tokens i in the current set $S = \{I_{k_0}, I_{k_1}, I_{k_2} ... I_{k_n}\}$ with any strings j in the result set $I_{k_{n+1}}$ which are superstrings of i and $support$(i) = $support$(j). This reduces redundancy of tokens such as “SIP" and “SIP/" when the sets are unioned together.

Certain tokens. Substrings with support equal to 1.0.

Frequent tokens. Substrings with support equal to a manually configured threshold. For example, in SIP 2.0 the method words `INVITE`, `ACK`, and `BYE` frequently occur in

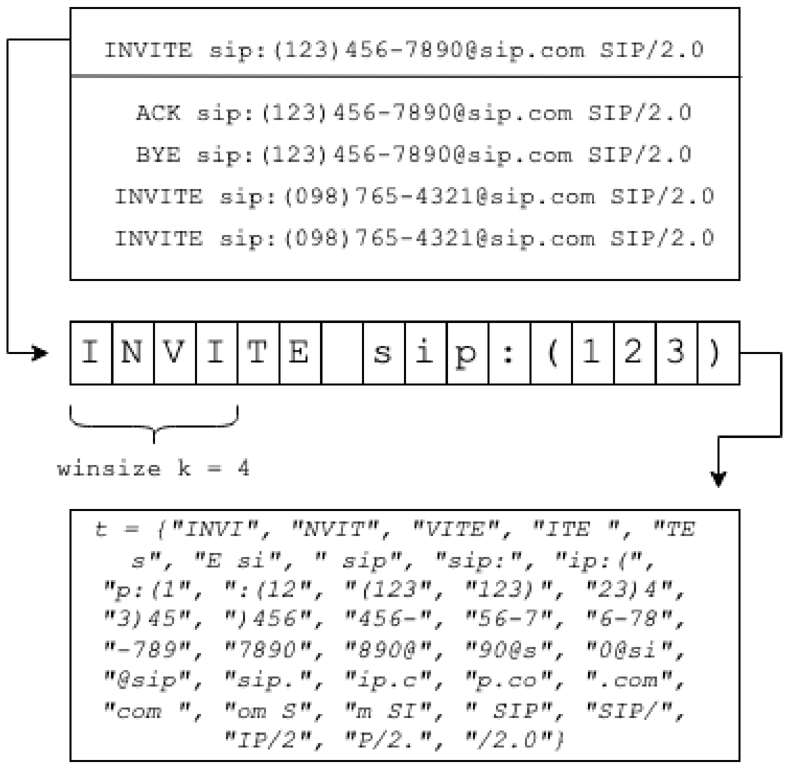

INVITE sip:(123)456-7890@sip.com SIP/2.0
ACK sip:(123)456-7890@sip.com SIP/2.0
BYE sip:(123)456-7890@sip.com SIP/2.0
INVITE sip:(098)765-4321@sip.com SIP/2.0
INVITE sip:(098)765-4321@sip.com SIP/2.0
I N V I T E s i p : (1 2 3)
winsize k = 4
t = {"INVI", "NVIT", "VITE", "ITE ", "TE s", "E si", " sip", "sip:", "ip:(", "p:(1", ":(12", "(123", "123)", "23)4", "3)45", ")456", "456-", "56-7", "6-78", "-789", "7890", "890@", "90@s", "0@si", "@sip", "sip.", "ip.c", "p.co", ".com", "com ", "om S", "m SI", " SIP", "SIP/", "IP/2", "P/2.", "/2.0"}

Figure 3.3: Modified Apriori Algorithm for string packet data using a sliding window algorithm. The sliding window process is repeated for all packet strings in the database file to make the full set of transactions.

request packets.

In order to give positional context to tokens, we introduce position frequency tables from natural language processing (NLP). This allows TRex to compute prefix and suffix groups from frequent tokens. Given a list of tokens $t_1, t_2, t_3, ...t_n$, TRex constructs a position frequency table for each token in each starting line in the database. The resulting vector $V = [\{t_1 : pos_1, t_2 : pos_2, t_3 : pos_3, ...t_n : pos_n\}, \{t_1 : pos_1, t_2 : pos_2, t_3 : pos_3, ...t_n : pos_n\}...]$ is referenced for tokens which occur at $pos = 0$. If the sum of support for those tokens found at $pos = 0$ is approximately 1.0 with respect to a minimal noise threshold, all tokens in this subset are captured in a prefix group. In the SIP request example, the methods `INVITE`, `ACK`, and `BYE` would be captured and separated with regular expression choice operators and a beginning position constraint in the following manner: ^(`INVITE`|`ACK`|`BYE`).

For suffixes, TRex uses string manipulation to reverse both the tokens and starting lines, which intuitively reverses the position significance also so that $pos = len(starting\_line)$ is now decides what goes into the capture group. Frequent tokens are reversed back to their original order before being inserted into the suffix, and choice operators and an end position constraint are appended. Example suffixes may include response codes, status codes or messages, or version numbers of protocols. An example output from TRex configured for HTTP requests and run on data containing both v. 1.0 and v. 1.1 requests may derive a suffix such as (`HTTP/1.0`|`HTTP/1.1`)$.

In addition to forming prefixes and suffixes, position frequency tables also give meaning to single-byte tokens which occur with $freq = 1.0$ at a fixed position. Wang et al found that some protocols have distinctive features at fixed offsets. As an example, they showed that the QQ messaging application always ends its packets with the byte `0x03`, indicating the end of a message [53]. We calculate position frequency tables similar to their approach for each character in each starting line and repeat the process in reverse in order to find single-byte tokens. These tokens are then added to the certain tokens set for future regular expression insertion. The substring tokens are then extracted from the relevant database lines and the remainder is preserved for the alignment stage.

3.3.2 GRex: Genetic Sequence Alignment of Common Substrings

GRex uses principles from genetic sequencing algorithms in order to pairwise align the remaining starting line data. Because we pre-process data into starting lines and TRex removes the found prefixes and suffixes to concentrate on only data which should be aligned, it is appropriate in our system to use a global alignment algorithm which attempts to align the entire sequence pair. We also use progressive alignment to sequentially align pairwise sequences to derive a resulting optimal alignment.

We chose to utilize the Needleman-Wunsch Algorithm for scoring and global alignment [60]. As a dynamic programming algorithm, Needleman-Wunsch utilizes a 2D scoring matrix which rewards matching characters and penalizes mismatches according to predefined parameters. Furthermore, a gap penalty is introduced to account for insertions/deletions (indels) in the string alignment. The principle of the algorithm is to greedily maximize the score of alignment per cell. Each cell in the matrix also contains a pointer value which points to the origin of the highest score and will aid in the final trace-back stage [61]. The score function F with gap score g, sequence $A = \{a_1, a_2, a_3, ...a_n\}$, and sequence $B = \{b_1, b_2, b_3, ...b_k\}$ is defined as follows:

$$F(i,j) = max \begin{cases} F(i-1, j-1) + score(a_i, b_j) \\ F(i-1, j) + g \\ F(i, j-1) + g \end{cases} \tag{3.1}$$

In the initialization phase, the first row and column are set as the gap score times the distance from the origin, which is the upper left most corner of the matrix. From here, the algorithm recursively computes the match score, horizontal gap score, and vertical gap score for each cell in the matrix. The match score is calculated from the preceding diagonal cell's score and the score of alignment of the two characters (match or mismatch). The horizontal gap score is the sum of the score to the left and the and the gap score. Similarly, the vertical gap score is the sum of the cell above and the gap score. The cell is set to the maximum of these values and the pointer to the cell (left, vertical, or diagonal) where the maximum came from. Once the matrix has been filled, the pointers are used to trace back through to find the highest-scoring alignment [61]. Figure 3.4 demonstrates this scoring algorithm in a matrix.

GRex modifies the Needleman-Wunsch algorithm to encode Greek symbols into our alignments which represent specific character classes, insertions and deletions, and mismatches which will later be decoded in the final translation step and used

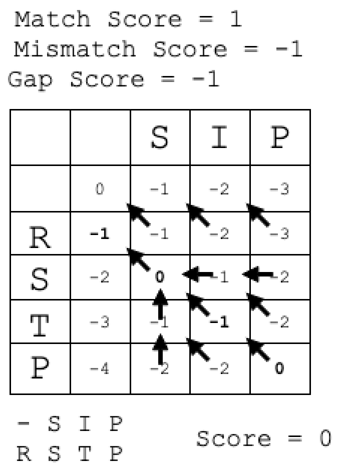


Figure 3.4: Scoring matrix example of the Needleman-Wunsch Algorithm. Traceback follows from the right bottom corner to the top left cell. The alignment results in the maximum possible score = 0.

Table 3.1: Greek Symbols For GRex Encoding

Φ	insertion/deletion
Δ	mismatch
ψ	gap
Σ	alphabet character
ω	not alphabet character
Π	digit
λ	not digit
Ω	white space character
σ	not white space character

to generate regular expressions. Table 3.1 provides a key to GRex's Greek symbol encoding. Previous work used an encoding algorithm for indels and mismatches [53], but we created our own encoding schema for GRex and expanded this to also incorporate defining character classes. Specifically, we encode symbols based on whether a character in a given position is or is not a white space character, alphabet character, or digit. Table 3.2 shows the regular expression operators GRex derives from these encodings at the translation step.

The encoding process runs in a progressive manner, such that two strings are aligned, compared, and encoded, and that return is then passed again and compared

Table 3.2: Supported Regular Expression Operators

Type	Symbol	Definition
character class	.	matches any character
	[]	matches any character in brackets
	[ˆ]	matches any character not in brackets
	\s	white space character
	\d	digit
	\w	word character
	\S	not white space character
	\D	not digit
	\W	not word character
quantifier	*	matches zero or more times
	?	matches at most one time
	+	matches one or more times
	$\{m, n\}$	matches between m and n times
	$\{m,\}$	matches m or more times
	$\{,n\}$	matches at most n times
other	ˆ	matches beginning of data
	$	matches end of data
	()	capture group
	|	OR operator

against another string in the sequence. Once aligned using the Needleman-Wunsch algorithm, the character types of each index are compared. Depending on the type of character, the returned string from this process is appended with a representation of either the character itself if the indexes are matching or a Greek character encoding otherwise. Once the new string is fully appended, it is returned to the loop. This returned string is compared to the next entry within the data. This process loops until every line from the packet data is aligned into the final encoding.

In order to create the regular expression, the replacement function is run against the generated string. Each index is checked and replaced with the regular expression counterpart with correct special character escaping. The final regular expression is constructed from the symbolic string. Frequent tokens are combined with choice operators and positional constraints ˆ and $ for prefixes and suffixes, respectively. Prefixes are appended to the beginning of the generated regular expression, and suffixes to the end.

3.3.3 HRex: High Performance Regular Expression Scanning

The final component is a sniffing application written in C++ which reads in PCAP files using `libpcap` and performs regular expression scanning with Intel's Hyperscan library [62]. Hyperscan provides significant performance improvement over other commercial but popular tools like Snort due to its reduction of duplicate operations in string and pattern matching and use of single-instruction, multiple data (SIMD) operations. In their experimentation, creators of Hyperscan found it outperformed Snort by a factor of 8.7 [63]. Our string literal tokens created by RExACtor's TRex can be applied in a pre-filter system like Snort, as well. Furthermore, Hyperscan does additionally support string literal, or keyword, matching. We provide both solutions for flexibility in real-world applications as the network capture environment is rarely one-size-fits-all. As our tokens are learned via unsupervised learning, they do not require manual analysis and thus scale well to larger, unknown data sets.

3.4 Experiments and Results

We performed two case studies using RExACtor regular expressions and tokens. The first is a comparison-based analysis using our own HTTP signature and ones developed by related works. Additionally, we assessed RExACtor's ability to develop a signature and keywords for SIP and RTSP, two protocols found in VoIP traffic flows. We use measures of precision, recall, and F1 score in order to assess signature efficacy.

We use measures of precision, recall, and F1 score in order to assess signature efficacy. Our measure of recall is the number of packets n correctly identified as protocol type P divided by the sum of the number of packets n correctly identified by their protocol type plus the number of packets m incorrectly identified as not P [29].

$$\text{Recall} = \frac{n \text{ correctly labeled } P}{n \text{ correctly labeled } P + m \text{ incorrectly labeled not } P}$$

Precision is defined as the number of packets n correctly identified as protocol type P divided by the sum of the number of packets n correctly identified as protocol type P plus the number of packets m incorrectly identified as P [29].

$$\text{Precision} = \frac{n \text{ correctly labeled } P}{n \text{ correctly labeled } P + m \text{ incorrectly labeled } P}$$

Finally, we also provide the F1 score for ours and other solutions.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

In addition to these data science measurements, we record, analyze, and compare heap usage using valgrind. We also compare the heap allocation for our signatures and signatures from comparable works to show our solution's improvements.

We use packet capture (PCAP) data from live network captures for both training and testing scenarios. An interesting note on training data is that in our cases, RExACtor did not require more than a few hundred starting lines in order to create quality signatures. We found that as long as the starting lines were from a diverse set of captures so that variable URLs, ports, and addresses were not over-represented, the common features were both discovered and generalized enough for re-usability. This demonstrates that RExACtor may be able to create specific signatures for protocols which may not appear frequently in a data set.

To create HTTP signatures with RExACtor, we used a small data set of mixed background traffic containing 213 HTTP request packets and 1000 HTTP responses. The resulting signatures and tokens are provided in Table 3.3. For the VoIP protocol tests, we used 252 SIP request packets and 230 responses for training signatures, and 638 RTSP responses and 562 RTSP requests. Our signatures and tokens for these are provided in Table 3.6. We ran tests on data sets from a live capture environment of background web traffic containing 1000 HTTP packets, 1000 SIP packets, 1000 RTSP packets, and 1000 packets of other mixed protocol types, including FTP, SMTP, ICMP, and others.

RExACtor is a Python tool library designed with Python 3.8. We use the efficient-apriori library (v. 1.1.1) [64] for tokenization, PyShark for data pre-processing (v. 0.4.3) [65], and numpy (v. 1.21) [66] data structures for optimization. The scanning framework used for testing regular expression matching is written in C++ using Boost 1.63 [67] and Hyperscan 5.4.0 [62] libraries for regular expression creation and matching, respectively. All experiments were performed on a single i9 Dell computer with Ubuntu 20.04 installed.

Table 3.3: HTTP Signatures and Tokens

Source	Side	Signatures	Tokens
AutoSig [56]	N/A	N/A	[HTTP/1.]; [GET\0x20/] [HTTP/1.];
LASER [55]	N/A	N/A	[HTTP/1.1]
Wang et al [53]	Request	^(GET | HEAD | POST).*HTTP/1.(1|0)\x0D\x0A	N/A
	Response	^HTTP/1.(1|0) [2|3|4|5]0.*\x0D\x0A	N/A
LCS Algorithm [57]	N/A	^(HTTP/1.1 | Host:.u | Connection:keep-alive | User-Agent:Mozilla/5.0 | Accept: | Accept-Encoding:ga | Accept-Language:en-US.*en;q=0.8 | Accept-Charset:ISO-8859-1)	[HTTP/1.1]; [Host:.u]; [Connection:keep-alive]; [User-Agent:Mozilla/5.0]; [Accept:]; [Accept-Encoding:ga]; [Accept-Language:en-US.*en;q=0.8]; [Accept-Charset:ISO-8859-1]
RExACtor	Request	^(GET / | POST / | POST | POST /mail | HEAD /).{0,2048}(HTTP/1.1\r\n)$	[HTTP/1.]
	Response	^(HTTP/1.1 |HTTP/1.).{0,3}.{0,24}\\S(\r\n)$	[HTTP/1.]

3.4.1 Case Study A: HTTP Detection

The results of scanning related works' regular expressions and our own on the testing dataset showed comparable recall for Wang et al's system and RExACtor. The longest common substring algorithm (LCS) signature performed significantly worse. For our dataset, all systems for both regular expression scanning and literal token scanning had perfect precision, including RExACtor. Because precision in our data set was ideal for all systems, the F1 Score results are comparable to the recall results.

While precision and recall results were on par with the state-of-the art, RExACtor's signature required significantly less heap space than either Wang et al's solution or LCS when performing regular expression scanning. This demonstrates RExACtor's superior performance in real embedded systems where memory optimization is

Table 3.6: VoIP Signatures and Tokens

Protocol	Side	Signatures	Tokens
SIP	Request	ˆ(INVITE sip:|ACK sip:|BYE sip:|REGISTER sip:).{0,74}(SIP/2.0)$	[sip:];[SIP/2.0]
	Response	ˆ(SIP/2.0 200 OK|SIP/2.0 40|SIP/2.0 100 Trying|SIP/2.0 180 Ringing|SIP/2.0 4|SIP/2.0 18|SIP/2.0).{0,31}	[SIP/2.0]
RTSP	Request	ˆ(TEARDOWN rtsp://|DESCRIBE rtsp://|SETUP rtsp://|PLAY rtsp://).{0,85}(RTSP/1.0\r\n)$	[rtsp://];[RTSP/1.0\r\n]
	Response	ˆ(RTSP/1.0)\\d\\d\\d .{0,6}[a-zA-Z](\r\n)$	[RTSP/1.0]

a critical factor.

Table 3.4: HTTP Case Study Results for Regular Expressions

Solution	P	R	F1	Allocation
Wang et al	1.000	0.984	0.992	20,783,011 bytes
LCS	1.000	0.254	0.405	36,015,510 bytes
RExACtor	1.000	0.976	0.989	18,310,656 bytes

Table 3.5: HTTP Case Study Results for Tokens

Solution	P	R	F1	Allocation
AutoSig	1.000	0.989	0.994	548,715 bytes
LASER	1.000	0.865	0.928	457,778 bytes
RExACtor	1.000	0.989	0.994	457,280 bytes

3.4.2 Case Study B: VoIP Detection

In addition to a comparison study using HTTP, we ran precision, recall, and memory allocation tests for RExACtor signatures for two VoIP protocols. In future work, we could expand this to chat, email, and other text-based protocols with the goal of targeting signatures for application-specific signature generation and protocol analysis. For example, we would want to target identifying and distinguishing between Skype, Zoom, or Webex traffic flows [29]. RExACtor performed amicably with perfect recall and precision in our dataset. Heap allocation results were also comparable to HTTP, showing that the good performance from the case study was not anomalous but rather indicative of good signature construction by RExACtor.

Table 3.7: VoIP Results for Regular Expressions

Protocol	P	R	F1	Allocation
SIP	1.000	1.000	1.000	34,449,057 bytes
RTSP	1.000	1.000	1.000	14,972,715 bytes

Table 3.8: VoIP Results for Tokens

Protocol	P	R	F1	Allocation
SIP	1.000	1.000	1.000	3,234,815 bytes
RTSP	1.000	1.000	1.000	3,624,176 bytes

RExACtor provides a classification solution for per-packet analysis and does not require data to be read as or reconstructed into flows in order to achieve classification. The modular design of RExACtor also allows for components to be used as tool sets so that engineers may adapt the knowledge discovery elements to their own scanning systems or use the full system as a whole framework. For example, the module TRex which derives tokens may be run individually in order to create keywords for Snort filters. Additionally, regular expression signatures can be created and published outside of the scanner in order to be plugged into any other DPI application. RExACtor also supports tokenization and alignment for raw data, allowing for analysis and signature creation for unknown protocols. This high adaptability means RExACtor can be applied in many network environments for knowledge discovery and DPI usage. Currently, RExACtor supports email (SMTP, POP3, IMAP), file transfer (FTP), chat (XMP, IRC), VoIP (SIP, RTSP), and web (HTTP) protocol analysis. Because RExACtor utilizes unsupervised learning and derives features from training data, it could be applied to learning more application specific signatures. One improvement for our solution and others discussed which use unsupervised learning would be an attempt to learn an optimal threshold for feature support rather than rely on manual configuration. In the comparison study, we found that packets not identified by the signatures were HTTP continuation packets of just data payload. Sessionization of protocol flows would allow a detection system to pick up on the first HTTP packet of an exchange with identifying method information, then record the 5-tuple flow information for further scanning. But this technique is not usable in per-packet analysis, and is a limitation of the problem space rather than our solution.

Regular expressions as a text-based approach also have limitations to their applicability in the real, complex network environment. The current industry standard for application layer DPI is to use regular expressions to match expected values in payloads. Protocols or applications may have multiple signatures to match request, response, or message types. Some protocols also do not have strong signatures. Furthermore, different versions of protocols or updated standards and RFCs often require separate signature sets and the continual update of older patterns [68]. Some protocols are also so non-standard that it is difficult to track them on a regular expression signature alone; for example, the payload of an RTP packet is raw data and assigned ports are dynamic [69]. Current state-of-the-art scanners suffer from variable and data-dependent performance impact [70]. With the increasing size and complexity of rulesets and the growing scale of the network environment, regular expression scanning using the current systems is more and more impractical. Regex methods are also limited by what classes they can be used for in network traffic. For example, there is no real regular expression signature which can identify Tor-routed traffic from non-Tor-routed traffic; the data is too diverse. Other desirable classes for traffic, such as traffic type, application, and user or device type present similar issues to regular expression scanners. Each possiblity must have its own, human-engineered regular expression to match precisely which is both slow and prone to error. In addition to scalability and machine performance, text-based regular expression matching no longer meets the needs of the environment as over 90% of browsing data today is encrypted [71].

Regular expression matching which relies on deterministic and non-deterministic finite automata (DFA and NFA) is prone to the state explosion problem, where computational complexity increases exponentially as regular expressions increase in number and become more complex. DFAs are fast to search, but are most prone to state explosion as every possibility is explicitly constructed. NFAs provide linear storage space to the size of the regular expression ruleset, but it is relatively easy to arrive at worst-case search performance with rule complexity [?]. [72]. Table 3.9 shows the processing complexity and storage costs of these structures where m is a number of regular expressions of length n.

Table 3.9: Worst-Case Space and Time Complexities for NFA and DFA [1]

Data Structure	**Complexity**	**Storage Cost**
NFA	$O(n^2m)$	$O(nm)$
DFA	$O(1)$	$O(\Sigma^{nm})$

Recently, hybrid NFA/DFA engines have been proposed to improve search performance; however, improvements on computational complexity can still be prone to memory issues in the case of large regular expression rulesets [62]. Parallel computing may alleviate some of the computational stress, but is acknowledged as a brute force approach [72]. Hyperscan is able to handle compiling and searching with incredible efficiency; however, this becomes exponentially worse to the point of impracticality when put at a test to scale, which we demonstrate in the next chapter of this work.

CHAPTER 4: ALPINE/PALM - SIMILARITY SEARCH THROUGH LOCALITY-SENSITIVE HASH FINGERPRINTING

4.1 Introduction

The ineffectiveness of existing regular expression DPI methods against encrypted payloads and weakly signatured data classes as well as the computational and storage complexity of automata has caused researchers to shift to other forms of traffic fingerprinting for identification. We propose using locality-sensitive hashing in order to measure similarity of features between packets. Locality-sensitive hashing uses a distance metric to preserve the similarity of original inputs when they are converted to locality-sensitive hashes (LSH). The benefit of locality-sensitive hashing here is that it is capable of reducing high-dimensionality data into a low-dimensional, space-effective, computationally efficient representation of the same data while still preserving similarity metrics between the data points [73]. Hash families may be defined for many distance measures such as Hamming distance, L1 and L2 norm [74], and Jaccard similarity [75].

Contrary to cryptographic hashes which attempt to avoid collision, locality-sensitive hashes preserve the similarity of data points (in our case, token sets) to one another. The following definition intuitively states that data points which are locally nearby have a higher probability of collision than further points.

Definition 1 *[2] A family H of functions from a domain S to a range U is called (r, ϵ, p_1, p_2)-sensitive, with r, $\epsilon > 0$, $p_1 > 0$, $p_2 > 0$, if for any p, $q \in S$, the following conditions hold:*

- *if $D(p, q) \leq r$ then $Pr_H[h(p) = h(q)] \geq p_1$,*

- *if* $D(p,q) > r(1+\epsilon)$ *then* $Pr_H[h(p) = h(q)] \leq p_2$.

One way in which locality-sensitive hashing has been applied in the network security domain is in user-level browser fingerprinting. In this method, web-based browser fingerprints created by extracting multiple values from the browser API may be hashed using MinHash or similar functions to generate signatures of high entropy, where the data is uniquely identifiable as a particular device or host [76]. We apply this concept similarly to network packet data, aiming for a locality-sensitive hash using features which should also evidence values unique to the class type. LSH clustering has also been used to create signatures for identifying malware at scale [77], but has rarely been applied in DPI.

Once hashes are created representative of the packets, differences in the hashes may be compared linearly using a similarity search approximation. Similarity search is a generalized term in data mining which refers to searching for objects where the available comparator is some common pattern or similarities among them. Examples of similarity search mechanisms include Nearest Neighbors searching, link-based similarity searches, duplicate detection, and defining object representations [2]. Applications for this in the cyber crime space include finding individuals in the same criminal network, finding content online similar to known evidence, querying to find images similar to another, and detecting fingerprints across network data. The best indexes for similarity search have efficient querying and storage mechanisms, minimum memory footprint, and minimal interference required by the developer and maintainer [78].

One of the challenges in data engineering is determining how to construct suitable, unique *tokens* which characterize the data meaningfully. The sentence, “Palm trees are native to the Pacific", may be split on whitespace into the set of tokens $T = \{$“palm",“trees",“are",“native",“to",“the",“pacific"$\}$. It is obvious that tokens like “the" are far too generic in the English language to be characteristic of this sentence, but a token like “palm" or “pacific" will be much more unique and indicate a stronger similarity. This idea may be more solidly quantified through taking the inverse document frequency (IDF) of tokens which minimizes the importance of terms which appear frequently in the document set. When combined with term frequency (TF) as the TF-IDF value [79], the most relevant tokens may be found which are both common and characteristically unique of the document set.

4.2 Previous Work

The Jaccard similarity of sets is commonly used to recommend products or next steps in a workflow, or detect plagiarism, or make predictions. These sets must be made up of individual items like the tokens or shingles of strings as done in SigBox [**?**] and RExACtor [68]. Both these works use the tokens as part of regular expression generation.

A limited amount of work has been done in applying locality-sensitive hashing in order to generate application fingerprints. Tang et al proposed HSLF [80], an HTTP header sequence-based LSH fingerprint generator for classifying applications in HTTP traffic. While this work is limited to only cleartext HTTP traffic, results show the ability of their SimHash-based method to accurately distinguish between data such as Firefox, VMWare, and WeChat. Naturally, this work has limitations due to encryption and the scope of traffic which only makes up a fraction of real-world data.

Jiang and Gokhale [81] show the ability of locality-sensitive hashing to accurately classify network traffic in their research using packet-level features exclusively in order to achieve stateless packet inspection. Their research focused solely on multimedia applications versus legacy web-browsing; they did not expand into encapsulation architectures and more traffic classes as well as application-level identification. They also use K-Nearest Neighbors clustering for classification, which is known to scale sub-optimally. Alpine and Palm expand these works into a much more complex and diverse set of experiments and applications.

4.3 Methodology

4.3.1 MinHash Algorithm and Locality-Sensitive Hash Forest

In order to reduce packet data to a set intersection problem, contents must first be *shingled* into items which will make up the set. An object is w-shingled when a sub-sequence of length w of contiguous tokens is cut from it. In ALPINE, we use the port information, transport layer protocol, flag values, and packet length to make up our shingles. In PALM, word tokens are created by delimiting packet payloads by whitespace. Thus, each original packet P is now associated with some set S_P of shingles as depicted in Figure 4.1.

$$J(P_A, P_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \tag{4.1}$$

Figure 4.1: Jaccard Similarity of Shingle Sets

MinHash may be used to quickly approximate the Jaccard similarity between two sets. The hash signatures take up a fixed and smaller amount of space than large shingle sets and can be used to normalize the data in cases where the shingle set features are of varying lengths and types. The MinHash of a given column is the number of the first row in permuted order whose characteristic matrix value is 1. This sequence is continued down the columns and repeated for k permutations. The probability that the MinHash function for a random permutation of rows produces the same value for two sets is a close approximation to their Jaccard similarity [82]. We use the LSHForest algorithm developed by Bawa et al [2] to index the locality-sensitive hashes and optimize the search process. It improves upon nearest-neighbor searches by both reducing complexity and eliminating the need to know the distance r from the query to the nearest neighbor of a given data point. This optimization is achieved through the use of a specific set of hash functions which ensures that any nearest neighbor query returns ϵ-approximate neighbors so long as a suitable l number of trees is chosen. Compared to previous work which uses nearest neighbor searches [81], LSHForest scales much more efficiently. The build time of a KNN model can have a complexity of $O(n^2)$, where n is the number of items. In contrast, LSH construction is done in linear time [2]. The theoretical complexities of operations on LSHForest are provided from the original paper in Table 4.1. When compared with time and space complexities for the automata in regular expression searching in Table 3.9, the logarithmic improvement and fixed storage space is clear.

Table 4.1: Time Complexities for LSHForest [2]

Operation	**Complexity**
Insertion	$O(l * log_B n)$
Deletion	$O(l * log_B n)$
Query	$O(l * log_B n) + O(l * log B) + O(M/B)$

In these equations, l represents the number of trees, B the branching factor of an internal node, and n the number of data points in the dataset. Storage is also optimized to be linear, $O(n)$, through the use of compressed PATRICIA tries [2]. The query for some point q for m nearest neighbors first performs a binary search for the longest matching prefix at a leaf node with the point s_i. Then some M points are collected synchronously across all trees and ranked by similarity score, with the top m being returned as an answer. We use the m number as votes to ultimately classify the sample by majority once the query is performed. This majority vote approach also allows for multiple classification results; for example, an adaptive system may be able to take the second closest result if the first choice turns out to be erroneous.

4.3.2 ALPINE: A Locality-Based Packet Inspection Engine

ALPINE [**?**] is **A L**ocality-Sensitive **P**acket **In**spection **E**ngine which performs shallow packet inspection on selected features from the network and transport layer encapsulation headers of packets. ALPINE uses Jaccard similarity in order to determine likeness between sets of the following packet header features: *source and destination IP address, type of service field, packet length, IP next protocol, IP flags, source and destination port* **??**. For indexing the computed MinHashes and performing lookup for classification, ALPINE uses the previously discussed LSHForest [2]. A query is performed by computing the MinHash of the input data and doing a binary search on the prefix trees. The hash is then mapped back to its label index. A majority vote is taken from the returned m nearest neighbors and the data is classified accordingly.

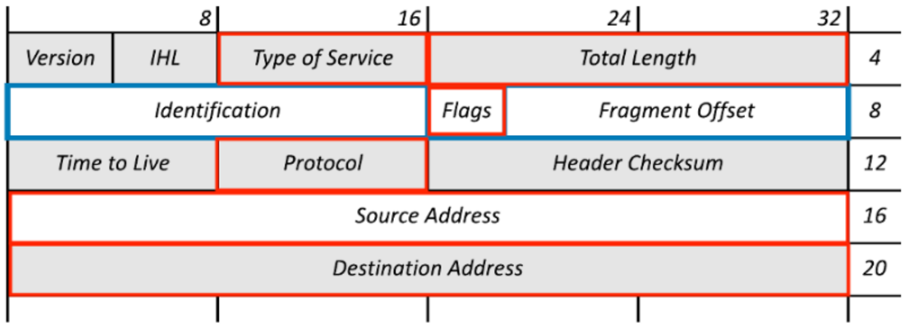


Figure 4.2: Example TCP/IPv4 header with extracted features for ALPINE highlighted in red.

4.3.3 PALM: Payload Analysis Using Locality Sensitive Measurements

PALM performs **P**ayload **A**nalysis using **L**ocality-Sensitive **M**easurements. This is a similar strategy to previous work in payload signature generation like SigBox [**?**] and RExACtor [68]. PALM delimits packet payloads by whitespace in order to create word-style tokens. In future work, other delimiters or a sliding-window shingling technique may suit identification of certain data types such as binary application layer protocols like HTTP2. In the experiments presented in this report, whitespace proved to be the most effective delimiter. We also experiment without using port numbers as identifiers in order to test the possiblity of non-standard port usage and detection. MinHash was used to generate the locality-sensitive hashes for the data samples and test samples. We use the LSHForest algorithm developed by Bawa et al [2] to index the locality-sensitive hashes and optimize the search process.

4.4 Experiments and Results

Table 4.2: Description of Combined Dataset

Application Type	**Protocol**	**# Samples**
File Transfer	FTP	10,015
	FTP-DATA	4,000
	BitTorrent	20,648
Voice-over-IP	MGCP	1,568
	SIP	1,112
	H.225	1,300
	RTP	15,552
	RTCP	1,626
Mail	SMTP	5,981
	POP	1,675
	IMAP	3,318
Authentication & Access	LDAP	1,354
	SMB	3,554
	Telnet	1,888
Tunneling	GPRS	9,981
	PPTP	1,288
	SSH	3,039
Web & Chat	DNS	10,563
	HTTP	21,298
	XMPP	1,553
	TLS	4000
Networking	DHCP	1,444
	NBNS	1,216
	GQUIC	1,740
	NTP	1,940
	SSDP	8,504

Instead of using a single dataset or network environment to evaluate the system, we used several available public datasets. Our sources include the CDX 2009 captures [83], the Skynet Tor dataset [84], the Canadian Institute for Cybersecurity's ISCX VPN/non-VPN and Tor/non-Tor datasets [85, 86], and repositories from Wireshark, Cloudshark, and IEEE Dataport. Investigation has shown that using captures from only a single environment can lead to bias in results [87]. A machine learning algorithm might learn characteristics of the network or environment such as IP addresses rather than actual protocol features, which can hurt generalizability of the model. For this reason and to increase the scope of classes beyond what is available in any single public dataset, we merged PCAPs from these experiments and created our own repository of twenty-six different, common application layer protocol packet captures. For experimental reproducibility and general use we have made this new dataset available publicly [1].

4.4.1 Protocol Auto Detection

The autodetection of application-layer protocols is a crucial functionality for lawful interception devices like DPI middleware boxes in order to properly assess, parse, and process the traffic they are checking. The traffic environment on any given signal can be wildly diverse; thus, any classification system must be capable of scaling to a large, multiclass problem. Previous systems [80, 81] have not attempted protocol detection to the scale of a 26-class problem, making this experiment and our model a true and fresh pioneer in the DPI field. To address the first question, we split the data into training and test sets and extracted the features from each of the protocols. We generated hash values for the training samples, adding them to the classifier. The results of the testing samples are provided in Table 4.3. The evidence shows that the combined model has high accuracy and precision/recall for protocol identifications. Cohen's kappa also shows a strong agreement among the ensemble voters. The method also performs equally well across application types, indicating generalizability. Figure 4.3 shows the confusion matrix results of the classifier's ensemble voting system.

4.4.2 Traffic Type

RFCs and standardized documentation necessitate that there are some detectable similarities between protocols. It is reasonable to think that detection of more abstract class types like traffic type (i.e. email, file transfer, web data) would be more

[1]https://github.com/mayakapoor/protocol-dataset

Table 4.3: Protocol Autodetection Results

	Jaccard			**TF-IDF**		
Accuracy	0.996			0.841		
Cohen's κ	0.997			0.841		
Protocol	**P**	**R**	**F1**	**P**	**R**	**F1**
BitTorrent	1.00	0.99	0.99	0.89	0.85	0.86
DHCP	1.00	1.00	1.00	0.91	1.00	0.95
DNS	1.00	0.99	1.00	0.95	0.97	0.96
FTP	0.99	1.00	0.99	0.83	0.90	0.86
FTP-DATA	1.00	1.00	1.00	0.93	0.96	0.94
GPRS	1.00	1.00	1.00	0.91	0.98	0.95
GQUIC	1.00	0.99	0.99	0.87	0.75	0.81
HTTP	1.00	0.99	0.99	0.82	0.48	0.61
H.225	1.00	1.00	1.00	0.83	0.97	0.90
IMAP	1.00	0.99	0.99	0.86	0.79	0.82
LDAP	1.00	0.99	0.99	0.87	1.00	0.93
MGCP	1.00	1.00	1.00	0.90	0.65	0.75
NBNS	0.99	1.00	0.99	0.94	1.00	0.97
NTP	1.00	1.00	1.00	0.91	1.00	0.95
POP	0.98	1.00	0.99	0.91	0.47	0.62
PPTP	0.95	1.00	0.98	0.84	0.90	0.87
RTCP	1.00	1.00	1.00	0.96	1.00	0.98
RTP	1.00	1.00	1.00	0.89	0.88	0.89
SIP	1.00	1.00	1.00	0.54	0.98	0.69
SMB	1.00	0.97	0.98	0.95	0.99	0.97
SMTP	1.00	0.99	0.99	0.92	0.93	0.93
SSDP	1.00	1.00	1.00	0.72	0.28	0.40
SSH	1.00	0.98	0.99	0.90	0.86	0.88
Telnet	0.99	1.00	0.99	0.86	0.99	0.92
TLS	1.00	1.00	1.00	1.00	1.00	1.00
XMPP	1.00	1.00	1.00	0.79	0.90	0.84

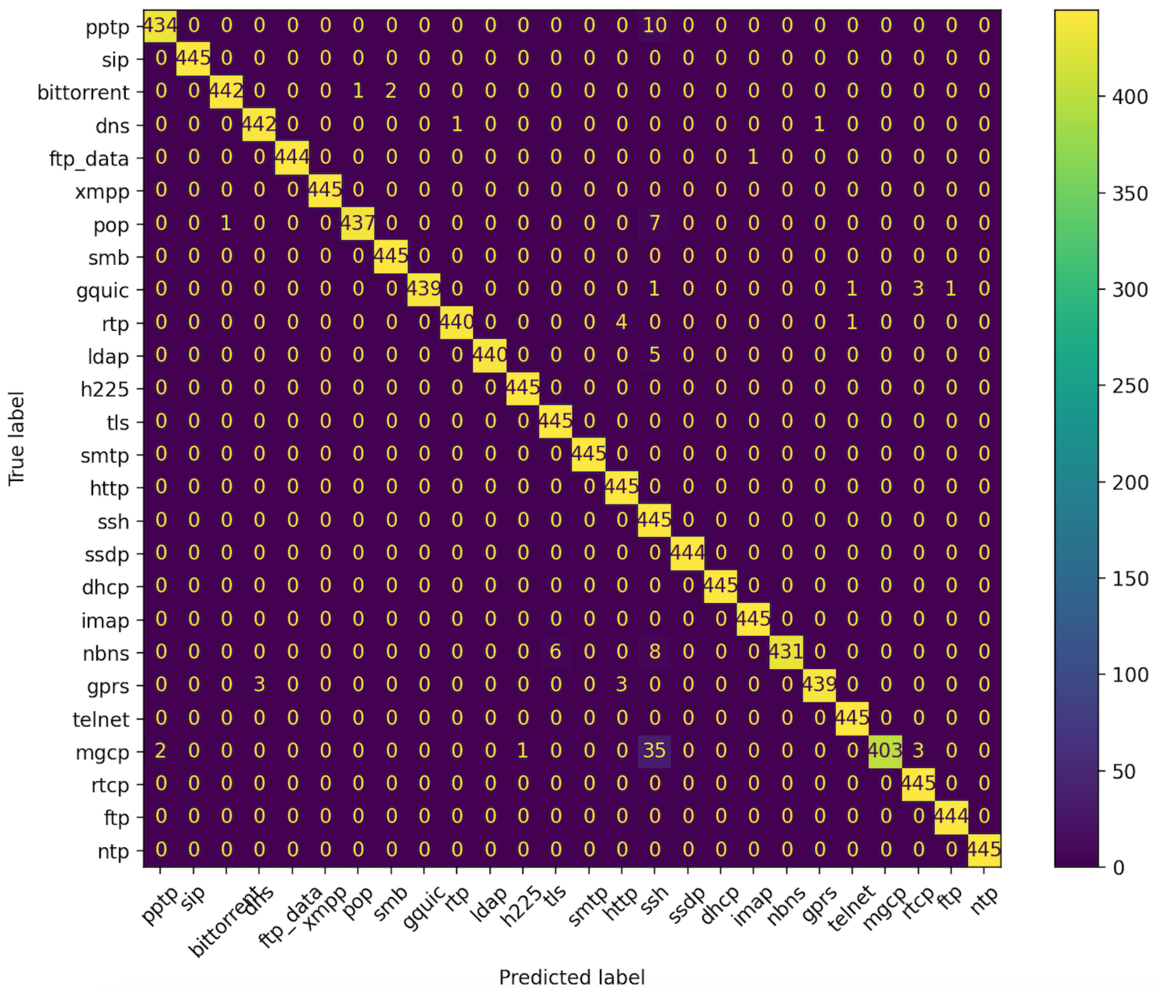


Figure 4.3: Twenty-six class identification problem using the ALPINE-PALM combined model.

heterogeneous and thus more difficult to embed. We experimented with classification by traffic type by amalgamating the dataset into classes as described in Table 4.2 in the datasets section of this work. Results in Figure 4.4 show a confusion matrix of the classification results which are also highly accurate.

4.4.3 TF-IDF Score Evaluation

In order to determine the effectivity of Jaccard similarity versus more complex similarity metrics, we evaluated the results of using payload tokens which were considered *high-value* by TF-IDF score. We posed this question to address concern that large data payloads, for example HTML documents or email messages, may introduce noise

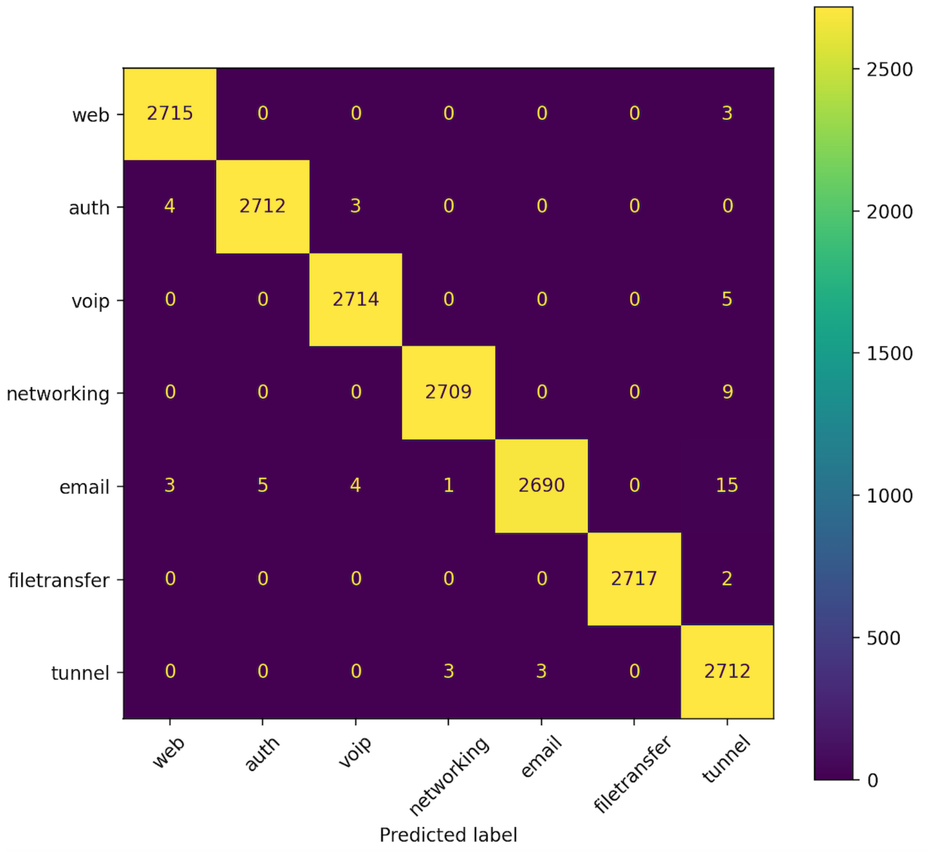


Figure 4.4: Confusion matrix of traffic type classification results.

into hashes which would reduce classification accuracy. Instead, isolating payload tokens to only the high value ones could reduce spurious extra data. In order to test this, we implemented a TF-IDF measurement of tokens in the training stage. This was done by modifying configurations of the PALM model to filter the added tokens. When training and test signatures were generated, only those tokens with TF-IDF score above the minimum threshold were kept and included in the actual LSH signature. The results in Table 4.3 show that this actually had a negative impact on the overall result, indicating the basic Jaccard similarity was a better distance metric for this use case.

Table 4.4: Multi-Hash Embedding Classification Results

	ALPINE			**PALM**			**Multi-Model**		
Accuracy	0.988			0.717			0.996		
Cohen's κ	0.988			0.705			0.997		
Protocol	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
BitTorrent	0.98	0.97	1.00	0.74	0.75	0.73	0.99	1.00	0.99
DHCP	1.00	1.00	1.00	0.91	0.83	1.00	1.00	1.00	1.00
DNS	1.00	1.00	1.00	0.24	0.46	0.17	1.00	1.00	0.99
FTP	0.96	0.97	0.94	0.90	0.83	0.98	0.99	0.99	1.00
FTPDATA	1.00	1.00	1.00	0.20	0.37	0.13	1.00	1.00	1.00
GPRS	1.00	1.00	1.00	0.91	0.98	0.85	1.00	1.00	1.00
GQUIC	0.99	0.99	1.00	0.27	0.48	0.19	0.99	1.00	0.98
HTTP	0.99	1.00	1.00	0.61	0.65	0.55	1.00	1.00	0.99
H.225	1.00	1.00	1.00	0.63	0.68	0.95	1.00	1.00	0.99
IMAP	0.99	0.94	0.98	0.85	0.77	0.59	0.99	1.00	0.99
LDAP	0.96	1.00	0.98	0.66	0.75	0.59	0.99	1.00	0.99
MGCP	1.00	1.00	1.00	0.90	0.83	1.00	1.00	1.00	1.00
NBNS	0.99	0.98	1.00	0.87	0.82	0.92	0.99	0.99	1.00
NTP	1.00	1.00	1.00	0.90	0.82	0.99	1.00	1.00	1.00
POP	0.99	0.98	0.99	0.26	0.41	0.20	0.99	0.98	1.00
PPTP	0.99	0.98	0.99	0.49	0.33	1.00	0.98	0.95	1.00
RTCP	1.00	1.00	1.00	0.92	0.85	1.00	1.00	1.00	1.00
RTP	0.99	0.98	0.99	0.40	0.65	0.29	1.00	1.00	1.00
SIP	1.00	1.00	1.00	0.93	0.87	1.00	1.00	1.00	1.00
SMB	0.99	1.00	0.98	0.81	0.80	0.82	0.98	1.00	0.97
SMTP	0.98	0.99	0.98	0.81	0.80	0.82	0.99	1.00	0.99
SSDP	0.99	1.00	0.98	0.92	0.86	1.00	1.00	1.00	1.00
SSH	0.95	0.97	0.92	0.47	0.67	0.37	0.99	1.00	0.98
Telnet	0.99	0.98	1.00	0.80	0.75	0.85	0.99	0.99	1.00
TLS	0.99	0.99	1.00	0.61	0.51	0.75	1.00	1.00	1.00
XMPP	1.00	1.00	0.99	0.92	1.00	0.85	1.00	1.00	1.00

4.4.4 LSH Multi-Embeddings

The concatenation of multiple classifiers and agreeance among voters in an ensemble has yielded a reduction in error and at best correlation when there is a misclassification [88]. We hypothesized that enabling both ALPINE and PALM to run together would yield the most accurate and agreed-upon classification results. In order to test this, we ran the 26-class protocol identification test against configurations of the model with just ALPINE embedding, only PALM embedding, and a final combined run. Our results as shown in Table 4.4 demonstrate that the combined votes of both models turned out to be the most accurate in the final tests. Furthermore, the agreeance among voters was also highest using multiple hash embeddings.

4.4.5 Model Performance and Throughput

One critical requirement for applied machine learning in network processing is that systems must keep up with signal processing at very high rates. Any passive system must not interfere with legitimate traffic and service. Furthermore, the software must be capable of performing the necessary processing on as much of the traffic as possible (ideally, all of it). While thinning and load-balancing solutions as well as diverting and copying traffic for offline analysis can help levy this concern, there is still the desire to employ classification solutions at real-time rates. Thus, we perform experiments to see how the system scales based on model sizes and number of classes. We measured the training time, system throughput in millibits per second (Mbps) as well as memory usage during the testing phase in megibytes (MiB). We ran all performance tests on the combined dataset consisting of 140,157 total packets. In order to avoid bias we implemented random under-sampling to even out the distribution of data, causing our total number of packets after sampling to be a fraction from the original set. For testing the number of classes, we created experiments with binary RTP/non-RTP classification, the traffic-type multiclass experiment with 8 classes/types, and the largest 26-class experiment where we identify all possible data layer protocols. In Table 4.6, we detail the results of the experiments for a varied number of classes and sample sizes.

Table 4.5: Performance Results for Varied Number of Classes

Classes	# Hashes	# Test Samples	Training Time	Test Time	Memory Usage
RTP/non-RTP	20,613	13,743	4.273s	1.21ms	470.8 MiB
Binary classification	2,400	1,600	0.738s	0.866ms	283.7 MiB
	240	160	0.075s	1.093ms	353.4 MiB
Traffic Type	28,543	19,029	7.136s	1.015ms	584.6 MiB
8-class problem	12,600	8,400	3.31s	1.006ms	305.3 MiB
	240	160	0.683s	2.397ms	290.6 MiB
Protocol	17,347	11,565	5.312s	1.138ms	352.4 MiB
26-class problem	15,600	10,400	3.13s	0.727ms	305.3 MiB
	1,560	1,040	0.492s	0.901ms	301.8 MiB

In order to test system scalability in deployment and verify the theoretical complexity discussed in the background section of this work, we performed a series of experiments increasing the number of signatures for a model used to classify HTTP traffic. As a comparator, we used the regular expression scanning tool built in RExACtor which uses the Hyperscan 5.4.0 regular expression matching library by Intel [62]. All experiments were performed on a single i9 Dell computer with Ubuntu 18.04 installed. In the trials, the number of signatures represented the number of LSH hashes our system or the number of regular expressions added to the sniffer application. We used RExACtor [68] to generate HTTP signatures from the HTTP traffic in our dataset. Both models performed the classification with 100% accuracy on all trials.

4.5 Discussion and Limitations

Using both models in parallel, we were able to assess the effectiveness of multi-embeddings of both the Alpine and Palm models. The concatenation of multiple classifiers and agreement among voters in an ensemble has yielded a reduction in error and at best correlation when there is a misclassification [88]. We hypothesized that enabling both Alpine and Palm to run together would yield the most accurate and

Table 4.6: Performance Results for Alpine and Palm as a multi-embedding versus Hyperscan

System	# Signatures	Training	Testing	Memory
ALPINE/PALM	1,000	1.21s	1.23s	392.4 MiB
	10,000	2.75s	3.56s	419.9 MiB
	100,000	52.75s	4.2s	1047.4 MiB
	1,000,000	423.3s	4.52s	4717.9 MiB
Hyperscan	1,000	4.6s	156ms	70.7 MiB
	10,000	81.8s	1.3s	529.3 MiB
	100,000	39.9m	17.28s	4596.7 MiB
	1,000,000	153m	63.1s	-

agreed-upon classification results. In order to test this, we ran the 26-class protocol identification test against configurations of the model with just Alpine embedding, only Palm embedding, and a final combined run. Our results as shown in Table 4.4 demonstrate that the combined votes of both models turned out to be the most accurate in the final tests. Furthermore, the agreeance among voters was also highest using multiple hash embeddings.

The locality-sensitive hashing approach is an improvement upon the regular expression approach to DPI in several ways. First, the hash generation process reduces the need for expert knowledge about a particular protocol. Previously, engineers would have to employ unique signatures for different message types. It would not be uncommon, for example, to have regular expression signatures for every method for a given text-based protocol like FTP or IMAP. Furthermore, requests and response messages often require unique signatures. Regexes are difficult for humans to read and comprehend. The management of server-client side interactions is also a challenge to know which versions to apply. Instead, hash embeddings using our forward-thinking system create a unique value for each provided input and indexes those values into one bucket. This even more generally captures the broad diversity of possible payload and header contents while providing a new layer of abstraction and simplicity. Flows may be examined bi-directionally, and all message types and methods considered.

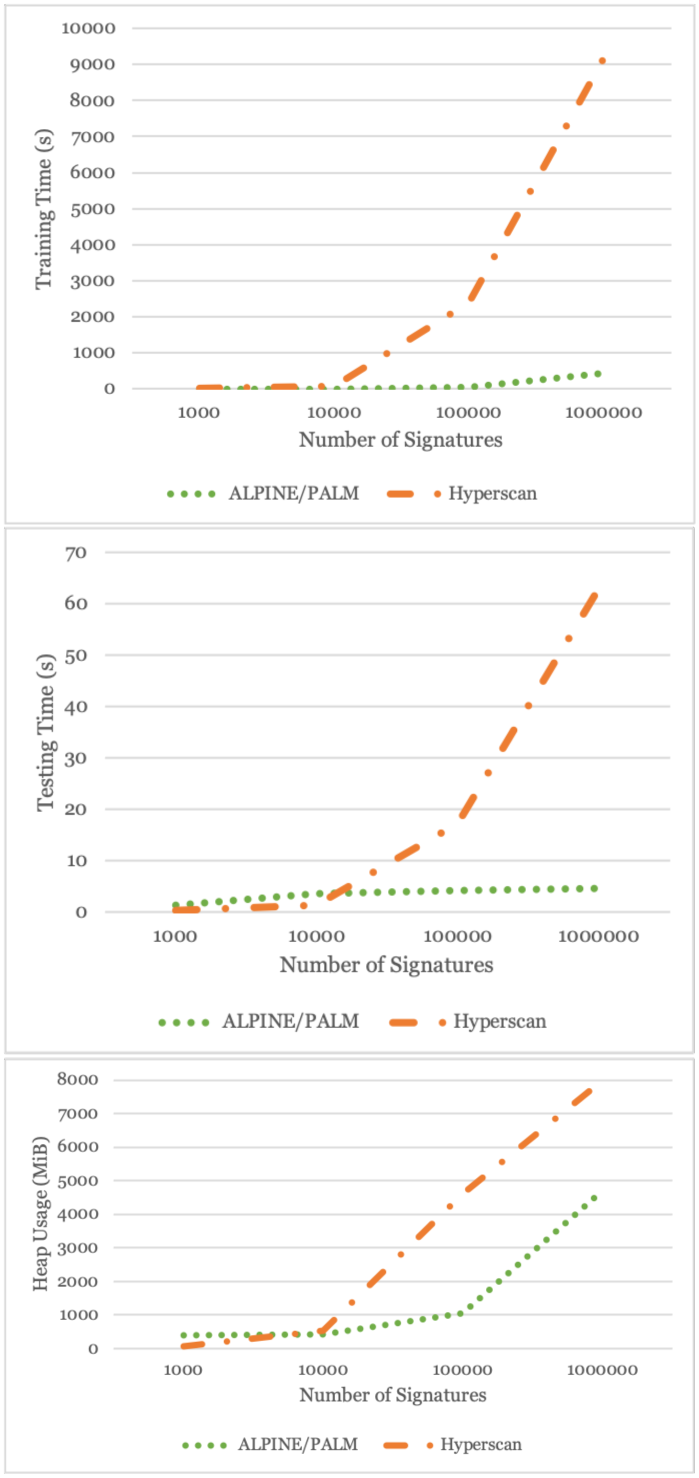


Figure 4.5: Time and space complexity results for ALPINE/PALM versus Hyperscan.

We are also not limited to traffic flows and protocol types. As demonstrated by the experiments in this paper, the system is capable of distinguishing more heterogenous or abstract class types like traffic type, mechanisms like encryption, and traditionally weak-signatured traffic like RTP. In terms of performance, the LSHForest as implemented in our model further reduces the storage and computational complexity when compared with a regular expression, automata-searching approach.

We posed the following research questions **R** to assess the ALPINE and PALM models and report our findings **F** here:

- **R1.** Given labeled training data, can packets be classified by their application-layer protocol through the construction and index of locality-sensitive hashes?

  F1. The results of our experiments show that for many application layer protocols, locality sensitive hashes may be constructed which are highly accurate for searching when indexed. Our model was 99.6% accurate in its identifications across a wide scale of protocols. In the real world, this sort of expandable model has applications in DPI technology for traffic surveillance, quality of service management and network management, traffic profiling, and content and copyright enforcement. There are also additional filtering applications for cybersecurity and private network management. Detecting traffic at the protocol level remains an important task for network engineers, and our hashing technique improves upon the state of the art in both accuracy and breadth of scope.

- **R2.** Can we also distinguish traffic types (such as email or VoIP data) from one another using this model? Does it generalize to other kinds of network traffic classes besides protocol?

  F2. Our experiment in classifying packets by traffic type tests both an additional definition of class for network traffic as well as the ability of the model to handle more heterogeneous classes. Even in this problem the hash embedding technique proved highly accurate. This demonstrates the generalizability of the models to real data and new problems.

- **R3.** For payload analysis, does using only high-value tokens as characterized by TF-IDF score improve distance measure results by isolating important features and reducing noise effectively in the data?

  F3. We bring to the forefront our metric choice of Jaccard similarity because machine learning for network traffic analysis has trended toward more and more technically advanced and often computationally intensive techniques to improve model accuracy or desired metrics. The results of our experiments in this report are evidence that a more complex metric or model is not always the best direction. Even within subfields of machine learning like natural language processing, there is not a silver bullet solution which works best for all kinds of problems. While TF-IDF decently diminishes tokens such as “a" and “the" from the English language, the fact that “HTTP\1.1" appears in nearly every HTTP request in our dataset is actually relevant considering the context. Especially in network traffic analysis where scalability and line-rate capability is highly desirable, simple techniques can still be effective and should not be disregarded.

- **R4.** Does the combination of the ALPINE and PALM models, i.e. multiple hash embeddings and concatenating those results, improve classification performance over any single hash embedding of a feature set?

  F4. Multi-embeddings are a proposed solution particularly for more heterogeneous datasets. The idea is that capturing multiple angles or representations of the data and combining those results will yield a more full-scope picture and analysis. Regular expressions, on the other hand, are one-dimensional and incapable of such encoding without becoming increasingly complex. Furthermore, a single bad bit can cause a packet match to fail. Network traffic and packets in the wild are extremely diverse and can be prone to error, and thus are an ideal test set for the multi-embedding hash technique which is both more informative and resilient. The modular design of our system enables using one or more embeddings, which allows for a balance between performance impact and accuracy. Interestingly, though the Palm model was overall poorer-performing than the Alpine model, votes from the Palm model helped improve the combined result regardless.

- **R5.** Does the number of classes, hashes, or sample size affect throughput or accuracy significantly?

  F5. One concern for applied machine learning in traffic analysis and cybersecurity in general is the performance impact such techniques have. Thus, we provide transparent numbers for our system performance. All experiments in this trial were run on a 1.6 GHz Dual-Core Intel Core i5 processor with 16GB DDR3 memory. In future work, leveraging an accelerated processor such as an FPGA would greatly enhance performance and throughput due to the repetitive nature of hash computations. In profiling, the most impactful procedure in the model is MinHash, which is likely optimizable through an FPGA [81]. The measure of what is acceptable throughput depends on the network environment; however, in terms of scalability it is valuable that our model does not seem to increase exponentially or vary in performance wildly depending on the data. Rather, it is data independent. Our model also trains quickly, meaning that models could be swapped or trained online in the field if needed. Another potential avenue for future research is batch optimization and parallel processing. It would be beneficial and possible to run some of the training computations in parallel. Furthermore, the distance measures could be taken in parallel for different models on separate threads and then the results aggregated.

- **R6.** Does our system outperform state-of-the-art regular expression scanning in terms of memory usage and testing/training time? What is the rate of scalability?

F6. As indicated in Table 4.6, for small signature sets Hyperscan initially outperformed our model; however, when scaled to larger signature sets, our model clearly is much more capable of handling the extra load. The comparative line graphs in Figure 4.5 illustrate the exponential versus sublinear growth patterns of each measure of performance, clearly demonstrating our systems' scalability over the state of the art. Our model takes just over 7 minutes to construct the LSH Forest with 1 million signatures, while the Hyperscan implementation takes over 2 hours. Even our testing time is much smaller than the Hyperscan test time at scale. For memory usage, our model also performs with reduced space complexity as the scale increases. At 1 million signatures, our model used just over 4GB of heap space, while Hyperscan crashed our setup due to exceeding 8GB at the same scale. If the problem set or data structure size is small, Hyperscan could be an optimal choice; however, our model is more generalizable to larger data sets and adaptable to a variety of problems. It would not be unreasonable to scale to several million signatures in a real surveillance or DPI system which ALPINE and PALM would be better suited for than the current capability of regular expression matching.

ALPINE and PALM show the potential for locality-sensitive measurements to consider multiple layers of features and are resilient to non-standard implementation. They normalize heterogeneous data into fixed-size, comparable space. These models as part of a DPI system have potential to scale both in performance and storage capacity. But, there are still problems in network classification that elude this method of feature extraction. For example, encrypted or compressed payloads make the text-based approach of PALM or regular expressions generated by RExACtor ineffective. Distinguishing encrypted traffic payloads from plaintext or compressed payloads is also a classification problem not appropriately dealt with through these methods. If ALPINE features of the header are not enough alone to distinguish the traffic type or particular application the traffic represents, then the classification will be inaccurate. If the header is encrypted, this also renders ALPINE ineffective. In further research, a method of understanding and profiling encrypted and compressed traffic or obfuscated payloads would greatly enhance deep learning-based DPI systems.

CHAPTER 5: MAPLE - IMAGE REPRESENTATION AND RECOGNITION OF INTERNET TRAFFIC

5.1 Introduction

In chapters 3 and 4, we discuss the difficulty of generating token-based or regular expression-based signatures for weakly signatured, variable data. This type of data floods modern internet traffic today; examples would be voice-over-IP (VoIP), streaming services, downloads and file transfers, and peer-to-peer sharing. Furthermore, encrypted traffic payloads or compressed data make generating signatures from payload text ineffective. The tokenization strategy of Palm is defeated by these techniques, and as Alpine focuses only on header features there is still the problem of an unanalyzed payload with potentially useful, untapped information.

In 2021, it was projected that 3 billion people use VoIP technology as a regular part of their daily lives. In 2021-2022, Zoom reported over 300 million users [89] Webex reported over 600 million, and Microsoft Teams recorded 275 million participants [90]. The widespread adaptation of VoIP technology has highlighted several cybersecurity vulnerabilities which make this type of traffic even more important to network intrusion detection systems. It is simple to spoof IP addresses or URIs with VoIP technology and produce robocalls used in phishing/spam or in denial of service attacks [91]. Malware can also be embedded into VoIP packet data, and is thus vulnerable to being the transport for backdoors, worms, trojans, and viruses [92, 93]. Cybersecurity specialists designing intrusion detection systems need to be able to intercept and process VoIP streams in order to detect these kinds of intrusions [94]. In the forensic environment, VoIP call analysis and reconstruction can provide critical evidence of criminal activity. Policy or content rights violations or copyright infringement may also be detected through the analysis of the reconstructed calls [95, 96]. This process and search capability is also useful to intelligence operations to find mission critical data from the enormous amount of traffic flowing in mission networks [97].

A majority of mainstream VoIP protocols use the real-time transport protocol (RTP) as their transport layer for encapsulation. The application layer protocol, for example the Session Initiation Protocol (SIP), Media Gateway Control Protocol (MGCP), or H.248, will establish the connection between endpoints and carry important session information such as codec encoding and metadata for the call. The application then relies on RTP to manage the dataflow between the connected systems. In the *complex, real network environment*, the implementation of SIP and RTP data flow is often sent de-coupled across physical signals and ports which presents a challenge for cybersecurity specialists using middlebox technologies for packet inspection and call reconstruction. Because the RTP data is encoded, it is necessary to know the signaling information in order to retrieve the codec information for proper decoding. Furthermore, the metadata associated with signaling protocols such as the SIP URI identifier provides necessary enrichment and context for the actual call. Lastly, RTP data may arrive at the middlebox before the signaling information which contains the port numbers associated with the RTP stream for that particular call; thus, the middlebox must retrospectively identify data packets (RTP) which belong to particular signaling information headers (SIP, for example).

Detecting the RTP stream itself from other types of traffic can be a difficult problem because the RTP signature is weak. Furthermore, the rise of encryption (SRTP) makes text-based signatures ineffective. The problem of accurately classifying network traffic has expanded beyond the scope of capability of text-based solutions. Instead, we propose using higher dimensionality of network traffic in order to advance RTP detection capability. MAPLE is a **ma**trix-based **p**ay**l**oad **e**ncoder which transforms packets into grayscale images to unveil hidden representation which may be used as input into a convolutional neural network model. Using this approach, we are able to expand detection capability to weakly signatured data, encrypted versus compressed traffic, and better traffic type and application profiling.

5.2 Previous Work

Convolutional neural networks are designed to process pixelated data and discover visual patterns in the latent space. CNNs organize their data into three dimensions, the spatial dimensionality of the input (height and width) and the depth.

The CNN model is comprised of three types of layers following the input layer. A *convolutional layer* uses a kernel filter in order to compute feature maps. The feature value of location (i, j) in the kth feature map of the ith layer $z^l_{i,j,k'}$ is calculated by:

$$z^l_{i,j,k} = w^{l^T}_k x^l_{i,j} + b^l_k \tag{5.1}$$

where w^l_k and b^l_k are the weight vector and bias term of the kth filter at the lth layer. The activation function, usually sigmoid, tanh, or rectified lineur unit (ReLU), may then be calculated as:

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{5.2}$$

A *pooling* layer performs down-sampling along the spatial dimensionality of the given input in order to reduce dimensions. For each feature map $a^l_{i,j,k}$, pooling can be calculated as follows:

$$y^l_{i,j,k} = pool(a^l_{m,n,k}), \forall(m, n) \in R_{ij} \tag{5.3}$$

where R_{ij} represents the local neighborhood around point (i, j). Common pooling functions include max pooling and average pooling [98]. Finally, *fully-connected* layer takes input from the previous layer and connects them to neurons of the output layer in order to perform high-level reasoning or learn some global semantic information [99].

LeNet [100] is a traditional CNN architecture for image recognition. It consists of two sets of convolutional, activation, and pooling layers. There is a final set of fully-connected, activation, and fully-connected layers with a softmax normalization and classification at the output layer.

Deep neural networks suffer from accuracy degradation due to how difficult it can become to map intermediary or residual layer outputs to inputs. A residual convolutional neural network (ResNet) is built up of blocks of stacked layers formed from a series of convolutions and non-linear activation functions. Following the authors who first introduced deep residual networks [101], we define a block as:

$$y = F(x, \{W_i\}) + x \tag{5.4}$$

Where x and y are the input and output vectors, respectively, and $F(x, \{W_i\})$ is the residual mapping with a set of weights W_i. If $H(x)$ represents the ideal output mapping that corresponds to the ground truth, residual networks hypothesize that $F(x) + x = H(x)$. The residual function is pushed to zero such that the equation becomes an identity function $H(x) = x$. This identity mapping, or shortcut layer, is what reduces the degradation problem and minimizes training error as the network becomes deeper [101].

CNNs have been used to classify network traffic in multiple problems for both per-packet and per-flow scenarios. Lim et al [102] also transform packets into grayscale images and label them by application (RDP, Skype, BitTorrent, and others). They use both a CNN and ResNet architecture. When using payloads of 1024B and a ResNet architecture, they achieved a 0.97 F1 score for accurately classifying the packets as one of eight types of applications.

Yang et al [**?**] use quantile normalization to transform network packets in CANs into pixelated images for CNN processing. They also take a flow-based strategy, combining multiple packets into a single image. For example, they propose taking 27 packets with 9 features each and creating a $9 \times 9 \times 3$ image, where three channels RGB (red, green, blue) are provided for a single pixel. This is then used as input into their CNN architecture.

Similarly, Gao [?] proposes a CNN-LSTM architecture which reduces false positive rates in his network intrusion detection system for cloud computing environments. Numerical and character attributes from the KDD CUP 99 test data set are first pre-processed into a 10×42 matrix, where window size $w = 10$ and the number of features in the data set is 42. The data set is transposed to an image dataset where each matrix W is mapped to a 10×42 grayscale image. If v_{ij} represents the value of a pixel in matrix/image W at position i, j, the values $v_{ij} \in [0, 255]$. To reduce the data interval and normalize values, the following formula is used:

$$x = \frac{x - \overline{x}}{\delta} \tag{5.5}$$

where x is the current value of a certain data in the sequence, $\overline{x}$ is the mean value of the data in the series, and δ is the standard deviation of the sequence. Finally, this system adds an additional attention mechanism layer to improve classification accuracy:

$$\begin{aligned} u_t &= tanh(W_w P_t + b_w), \\ a_t softmax&(u_t^T, u_w), \\ v &= \sum a_t P - T, \end{aligned} \tag{5.6}$$

where u_t is the attribute representation of P_t, W_w is the context vector randomly generated during training, a_t is the importance weight, and v is the high-level representation obtained through importance weight summation on P_t.

Jo et al [3] use a CNN-LSTM hybrid model to perform network intrusion detection, but first pre-process packets into image representations. Namely, they adapt the NSL-KDD dataset which has 41 fields into pixel representations. Continuous fields are first normalized to a $[0-255]$ range to create a grayscale image pixel value. Symbolic fields such as “protocol type" are represented by the same kind of pixel value, but normalized based on the number of possibilities. For example, a symbolic value with three possibilities will be normalized as 0, 128, or 255 [3]. 28×28 images are created and processed with a 5×5 kernel as shown in Figure 5.1.

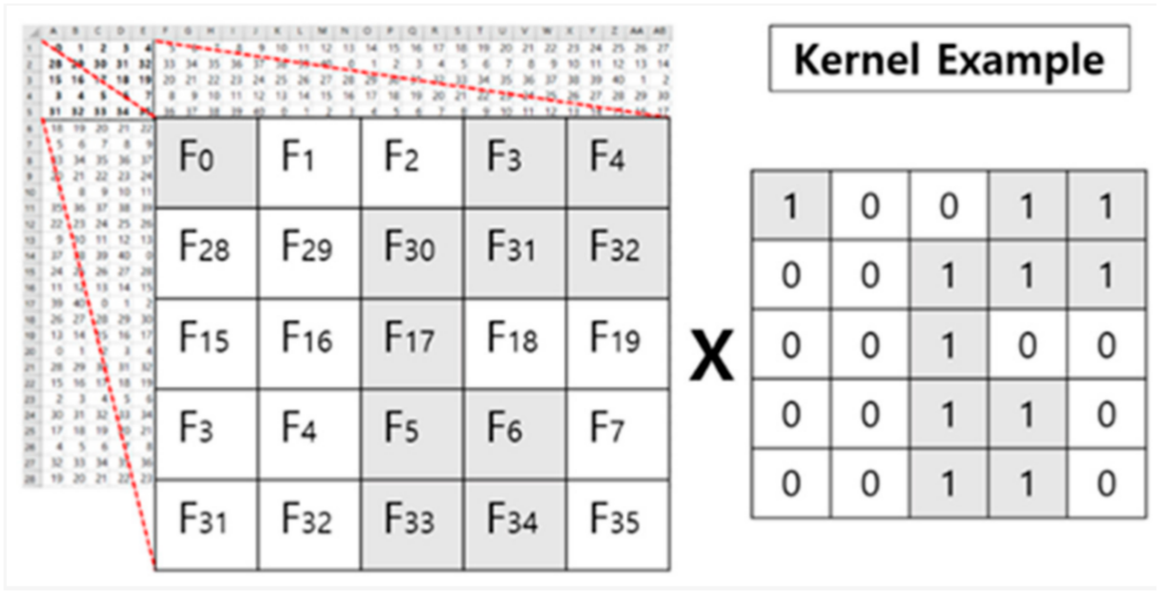


Figure 5.1: Example application of kernel to pixelated packet data [3]

Deep Packet [?] is a well-cited work in deep learning for traffic classification. They use a stacked auto-encoder and CNN to perform traffic classification into traffic types and application types as labeled in the VPN/non-VPN UNB ISCX dataset [85]. Their solution achieves 94% accuracy in the traffic classification task per traffic type, and 97% accuracy in the application classification task. Incorporating the same dataset as well as additional, more recent flows, we are able to achieve higher accuracy in RTP detection with a less computationally complex framework in Maple.

5.3 Methodology

It is expected that even in highly entropic or variable packet data, there will be some level of standardization (i.e. method names or status codes) or some commonalities such as user names, device details, semantic contexts, and other conversation or implementation-specific indicators. In order to delve deeper into these hidden layers and capture more latent space representations in packets, we propose in Maple to transform packets into grayscale images, hypothesizing that packets will appear similar to one another once transformed into their image representations. We base this assumption on previous work [102] which has shown similarities in grayscale images rendered from packet data in order to classify traffic by application and protocol type which are detectable by CNNs. Figure 5.2 asserts the validity of this assumption by showing comparative images rendered from packets in our combined dataset previously used and explained in detail in chapter 4 of this work.

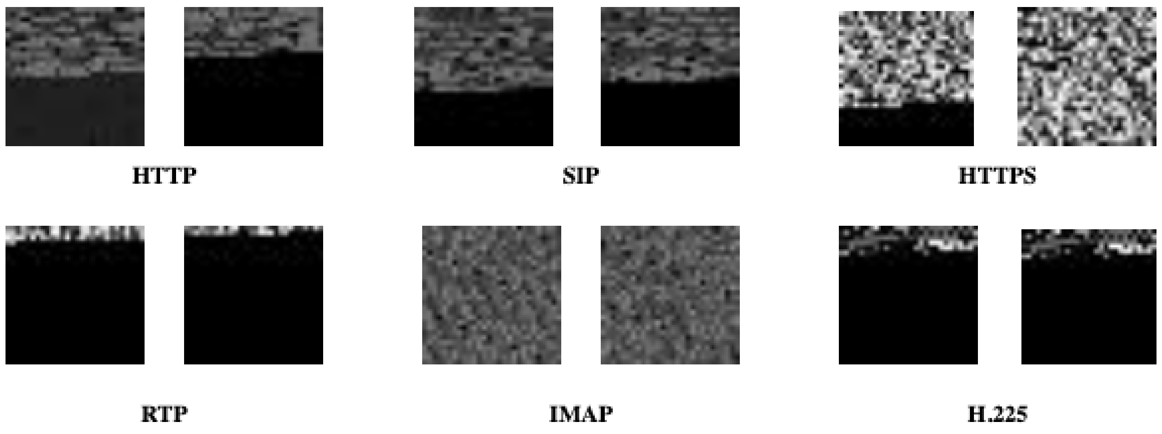


Figure 5.2: Grayscale images generated from packets in our combined dataset used in the following experiments.

To shape packet payloads into an image, we extract the payloads and convert them from byte encoding to a normalized decimal integer $i \in [0, 255]$ [3]. For image size, we chose a 28×28 representation, for a total of 784 input features. In cases where the payload is shorter than 784 bytes, we apply padding for input normalization, or otherwise truncate the payload data.

5.3.1 LeNet Model Configuration

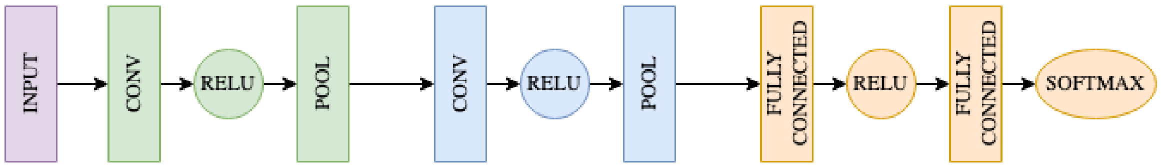


Figure 5.3: The architecture of our LeNet models.

We selected LeNet as a standard model in order to test the efficacy of convolutional neural networks to the RTP detection problem. LeNet has a low complexity, so has higher potential for practical use in real-time, line-rate packet inspection systems than deeper models which require more time. Figure 5.3 shows the set up of the model which we use as a baseline. We designed two separate models where one employed 6 and 16 kernels per convolutional layer (A) and another that used a 16 and 32 kernels (B). Each model contained two fully-connected, dense layers of size 1024 with a binary softmax classifier at the output layer.

5.3.2 ResNet Model Configuration

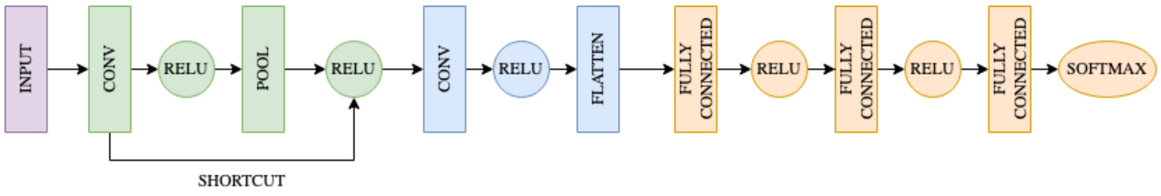


Figure 5.4: The architecture of our ResNet models.

For deployment purposes in real network environments, it is imperative that artificial intelligence solutions are capable of scaling to line-rate while still being able to perform the classification task to the required level of accuracy. While the definition of line-rate varies per network environment, the identity mappings of ResNet do not introduce additional complexity [101]. Thus, we introduce residual mapping as a potential solution to adding additional layers of representation and thus improve classification through more hidden features, while also minimizing overhead.

We developed four ResNet-based models for the MAPLE system's experiments. The first model (C) consists of three convolutional layers with 32, 32, and 64 kernels of dimension 3×3, and three dense layers with output sizes of 64, 32, and 2 respectively. The second model (D) follows a similar architecture except the values are halved. Convolutional layers had 16, 16, and 32 kernels per layer and the three dense layers had output sizes of 32, 16, and 2. Model (E) corresponds to the same number of kernels and output sizes as (C), except it uses a kernel dimension of 7×7. For all models, the adam optimizer was used as well as ReLU activation between all layers except the final, which used softmax for normalization and classification. Categorical cross-entropy was used as the loss function for training. We also employed a dropout layer in order to reduce model overfitting.

Table 5.1: Configurations of each model used in the experiments.

Type	Model	# Kernels	Kernel Size	FC Dim
LeNet	A	3×3	6, 16	1024, 2
	B	3×3	16, 32	1024, 2
ResNet	C	3×3	32, 32, 64	64, 32, 2
	D	3×3	16, 16, 32	32, 16, 2
	E	7×7	32, 32, 64	64, 32, 2

5.4 Experiments and Results

We ran several experiments to test the model's ability for binary RTP/non-RTP detection as well as multi-class protocol identification. All tests were performed on a single CPU of a 1.6 GHz dual-core Intel i5 processor with 16 GB DDR3 RAM. In the binary confusion matrix, true positive (TP) indicates correct classification of data. True negative (TN) is correct classification of data as non-RTP. false negative (FN) implies incorrect identification of traffic as non-RTP, and false positive (FP) is the incorrect classification of data as RTP. We use measurements of precision, recall, and F1-score for model evaluation, defined as follows:

$$Recall = \frac{TP}{TP+FP} \; Precision = \frac{TP}{TP+FN} \quad F1\ Score = \frac{2(P \times R)}{P+R} \tag{5.7}$$

5.4.1 RTP Detection

For the binary RTP/non-RTP classification, we re-labeled all non-RTP and non-SRTP traffic as *non-RTP*, and merged SRTP and RTP traffic into an *RTP* label. For testing the different configurations of the MAPLE model, traffic was first processed and transformed into a matrix image, and then used as input for each of the models. We performed random under-sampling to avoid bias in the dataset, and then split the data into 60%/40% for training and testing. First, in order to establish a baseline we ran the multi-model using ALPINE and PALM with the same datasets with the results in Table 6.1. We ran tests with training for 3 epochs for each CNN model. As we plan to deploy this technology in a real RTP detection and deep packet inspection system, we also measured classification throughput in order to determine how much traffic the MAPLE system would be able to process given the detection model configuration. Results are provided in Table 5.3.

Table 5.2: Classification results of RTP vs non-RTP detection for the LSH model.

Class	P	R	F1
Non-RTP	0.73	0.77	0.75
RTP	0.76	0.71	0.73

Generally, the ResNet model performed exceedingly well at the RTP detection task, exceeding ninety-nine percent accuracy across all the configurations. The LeNet models performed marginally worse in terms of accuracy, but the smaller of the two models had the highest throughput of any of the tested configurations. Another configuration choice which may have significant impact on model accuracy is the

Table 5.3: Classification results of RTP vs non-RTP detection for all Maple models.

Model	P	R	F1
A			
Non-RTP	0.96	0.99	0.98
RTP	0.99	0.96	0.98
B			
Non-RTP	0.97	0.99	0.98
RTP	0.99	0.97	0.98
C			
Non-RTP	1.00	0.99	1.00
RTP	0.99	1.00	1.00
D			
Non-RTP	1.00	1.00	1.00
RTP	1.00	1.00	1.00
E			
Non-RTP	1.00	1.00	1.00
RTP	1.00	1.00	1.00

Table 5.4: Throughput results of RTP vs non-RTP detection for all Maple models.

Model	Accuracy	Mb/s
A	0.975348	12.5
B	0.978258	9.28
C	0.996934	7.07
D	0.995908	10.41
E	0.996847	5.3

number of epochs trained. Real systems place an emphasis on minimizing down time of a system, but in actual deployment many models may be trained offline and then embedded. In systems where training time is not a factor of performance, we can thus increase the number of epochs with little real impact. In order to test the efficacy of such a design choice, we re-ran Maple models A and C with 10 epochs to see if performance improved. Results in Tables 5.5 and 5.6 show that ResNet appears to learn faster than LeNet, as it gained very little improvement with additional training time. Interestingly, with additional training time LeNet approaches similar accuracy to ResNet but still maintains higher throughput.

Table 5.5: Classification results of RTP vs non-RTP detection for Maple models A and C with additional training time.

Model	P	R	F1
A			
Non-RTP	0.99	0.99	0.99
RTP	0.99	0.99	0.99
C			
Non-RTP	1.00	1.00	1.00
RTP	1.00	1.00	1.00

Table 5.6: Throughput results of RTP vs non-RTP detection for Maple models A and C with additional training time.

Model	Accuracy	Mb/s
A	0.990499	14.99
C	0.998172	7.38

5.4.2 Protocol Auto Detection

As previously mentioned, many middlebox technologies are interested in multi-classification problems where many different applications or protocols need to be identified. Thus, we expand our experiments to a 26-protocol problem using the combined dataset described in chapter 4. We again use models A and C, LeNet and ResNet respectively, for this expansion. As this is a much harder problem, we expect to need more training time. We ran each model with 10 epochs and 50 epochs, and recorded the macro averages in Table 5.7.

Table 5.7: Results of multi-class detection for models A and C as macro averages.

Model	P	R	F1	Accuracy
A - 10 epochs	0.78	0.78	0.77	0.780920
50 epochs	0.83	0.83	0.83	0.832446
C - 10 epochs	0.84	0.84	0.83	0.835993
50 epochs	0.86	0.85	0.85	0.849528

Increasing the training time for the LeNet model showed significant improvement in accuracy, but not likely to a level needed for deployed systems so there is a need to build upon the baseline. The ResNet model approaches better accuracy with more training time.

5.4.3 Model Performance and Throughput

Decreasing the number of kernels may have slightly impacted accuracy, but improved overall throughput significantly and lessens memory footprint. In a real network environment, it may be worth consideration to sacrifice some accuracy in order to be able to monitor more traffic or more effectively load-balance the received input depending on hardware capability. Thus, the ideal model configuration is often environment-dependent and inspired us to incorporate configuration capability in Maple so that the deployed system may best suit the environment. We provide the throughput of each model configuration in our experiments in Table 5.4.

5.5 Discussion and Limitations

Maple works well on input data such as a payload which may be distinct from one data sample from another, but can be divided and matricized into units for comparison (i.e. turned into a grayscale image of uniform dimension). For deep packet inspection, this model is able to create latent representations of payload data which uniquely identify traffic of different types at a higher dimension than is considered by current signature-matching or filter-based solutions used in industry. Our implementation for this initial deployment solves the RTP detection problem with high accuracy using a minimal framework ideal for line-rate. The system could be further optimized by running these models in parallel; as the solution relies on per-packet analysis, throughput could be increased across models. One limitation of the proposed encoding is that individual packets must be large enough to create a significant image. Thus, this model would not work well on short packets or packets with no payload data. In this case, we propose using a combination of header features and payload when available would be a wiser strategy.

CHAPTER 6: DATE - POINT CLOUD REPRESENTATION AND DENSITY CLUSTERING ANALYSIS OF PACKETS

6.1 Introduction

In geometric topology, point clouds are another way to generate data shapes for anomaly detection and comparison. We propose DATE, a cluster **d**ensity **a**nalysis-based **t**ensor **e**ncoder model which expands network packets into three dimensional point clouds, which we then extract features of regarding cluster density in order to find commonalities and classify based on data shape. This spatial expansion is a novel application of latent representation learning to network traffic and the problem of RTP detection or detecting weakly signatured traffic and could expand to other protocol problems.

6.2 Previous Work

DATE is a novel contribution to the field of packet processing as there is little published, previous work in generating 3D point cloud representations of packets. Raw packet data from LiDAR (light detection and radiation) systems have been shaped losslessly into 2D matrices and point clouds [103]. This work introduces the problem of spatial correlation in packet data, namely that packets in their raw state are not usually uniform or in a state which may be processed using the spatial or geometric strategies employed by image and computer vision algorithms. Thus, data compression or pre-processing must be performed to create the necessary uniformity.

Costeux et al [104] worked on the fast detection of Skype and other RTP-based telephony traffic. They specify several fields from the RTP and encapsulation header which can be used to filter out non-RTP traffic. We employ this technique in the middlebox as an initial filtering step in our end-to-end data processing pipeline.

Kmet et al [95] build on Costeux by incorporating additional header features for per-packet selection, and adding a flow-based solution to reduce the over-selection problem. They were able to minimize mis-classification to nearly zero by buffering up to 10 packets of the RTP stream. The synchronization source number from the header is used along with timestamps to check for proper increment and stream correlation. We focus on per-packet identification only.

Some early research [105] describes expanding NetFlow data generated by routers into 2D manifolds. We reference this as an intial introduction of spatial expansion of traffic data in order to perform dimensionality reduction, which is a key theoretical linkage to our work.

6.3 Methodology

Date describes the density-based clustering analysis which is a novel contribution of this work; we also present Date in the context of a larger packet processing system for real AI applications.

6.3.1 Point Cloud Creation

Point clouds are a set of Cartesian coordinates (x, y, z) which represent some points in a three-dimensional space. In computer vision, point clouds may be used as a method for mapping high-dimensional shapes into lower-dimensioned space [106]. This latent representation is often used in a temporal sense to express the movement of objects in three-dimensional space. For example, a car may be modeled as a 3D point cloud, and its movement in space described as a series of clouds $c_0, c_1, ..., c_t$ where t represents a number of time intervals and c_n the point cloud generated at a given time slice. Unlike time-series based approaches which would require traffic flow data, we emulate the work of Quach et al [107] and treat the point clouds as static, analagous to manifolds in three dimensional space. This captures a geometrical representation of packet data which we use for cluster analysis.

In the DATE model, payload data is extracted from the packets and truncated to 1024 bytes. In cases where the payload is shorter than 1024, we add padding to normalize the input size. We convert the byte values v to decimal values $v' \in [0, 255]$. A sliding window technique is then used to create three-dimensional points in our feature space. We map each byte to a value in the (x, y, z) coordinate. The coordinates are mapped to a three-dimensional space to form point clouds like Figure 6.1. We expect that packets with similar data will thus result in similar point clouds.

For example, the byte string, '0x68', '0x65', '0x6c', '0x6c', '0x6f', would first be transformed to decimal values (104, 101, 108, 108, 111). Then, the sliding window would produce the points corresponding to coordinates

$$\begin{aligned} C = \{&(104, 101, 108), \\ &(101, 108, 108), \\ &(108, 108, 111)\}. \end{aligned} \tag{6.1}$$

6.3.2 Density-Based Spatial Clustering of Applications With Noise (DBSCAN)

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) uses three different point classifications (see Figure 6.2): *Core points* represent centers of clusters and points are labeled as such based on the number of neighbors they have compared to their neighbors. *Border points* represent the edges and possess the least number of neighbors but are still within the range of a core point. Noise or *Outlier points* are neither borders nor cores and do not belong to a cluster. The DBSCAN algorithm visits each point and classifies them as such in order to paint a picture of where clusters of data are [108].

After the point clouds are created, we process them using the DBSCAN algorithm in order to determine natural clusters that exist in the provided classes in the ground

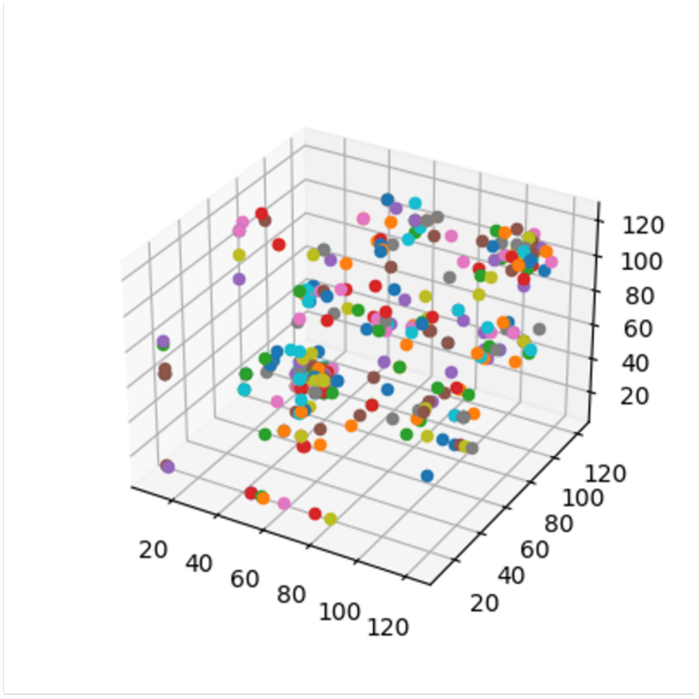


Figure 6.1: 3D point cloud made out of a Session Initiation Protocol (SIP) packet.

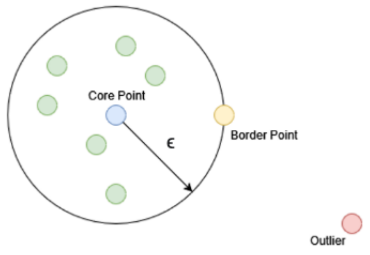


Figure 6.2: Illustration of types of points in DBSCAN.

truth. DBSCAN works by grouping points that are close together utilizing a Euclidean distance for measurements. Thresholds exist in the algorithm to specify the minimum number of points needed to create a cluster and the value of the Euclidean distance that determines if points are close enough to make a cluster. We configure two parameters in the algorithm: ϵ and a minimum number of samples. The ϵ value is the radius for each circle drawn around each point to query density, and the minimum number of samples is the minimum number of points required to be inside the drawn circle for it to be considered a Core point.

The minimum number of samples must at least be the number of dimensions, and as a heuristic, it is generally twice the dimensions. All tests were performed with the DBSCAN configuration of $\epsilon = 7$ and `min_samples` $= 7$. Having a baseline of 7 points to make a cluster within a 3D dimensional plane would be forgiving enough to create more clusters in a lower populated point cloud. An ϵ value of 7 should also allow for a greater number of clusters to be recognized [109].

Tuning DBSCAN has the potential to uncover great performance when it comes to creating recognizable clouds within the 3D space. As stated earlier, two inputs to DBSCAN can be tuned: ϵ and `min_samples`. It is important to have these values set reasonably to see favorable results. As a rule of thumb, the minimum number of samples needed to classify a cluster should be set to at least the number of dimensions. If it is set below this threshold, then singular points or two points that are close together would constitute a cluster, which would not be an accurate way of determining where data is clumped [110].

Similarly, ϵ needs to be set based on general rules. If the value of ϵ chosen is too small then more clusters will be created, and more data points will be taken as noise. However, if set too large, then a number of smaller clusters will be more likely to merge into one large cluster. One way that a value of ϵ can be estimated is by examining the input data and finding the average distance between each point and its k nearest neighbors. Because each packet is highly variable, choosing an exact value for ϵ is a difficult problem. An estimation of ϵ must be taken from a conglomeration of the packet's k-nearest neighbors data where k is the value of `min_samples` chosen. As shown in the figure above, the point of each line with the greatest curvature is considered the ideal value for ϵ. A line has been illustrated in Figure 6.3 where an estimated average is made for every sample. The variation in an ideal ϵ value could be a major factor in why some packets are recognizable in their cloud state and others are not [109].

We extract the following features as the feature vector for classification from the

DBSCAN results:

- clusterCount = number of clusters
- averageClusterSize = average cluster size
- standardDeviation = standard deviation
- noisePercent = percent of cloud containing noise

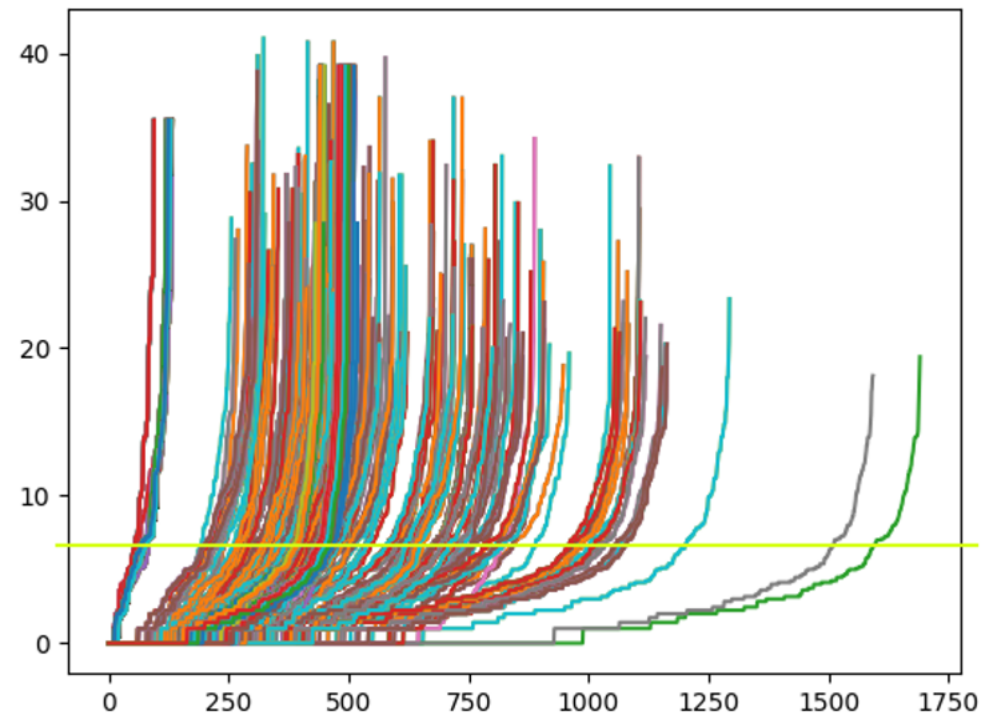


Figure 6.3: K-nearest neighbors graph generated for packets in the dataset using a `min_samples` = 6.

6.3.3 Packet Classification

For deep learning, we feed the extracted cluster features forward into a two-layer multilayer perceptron unit (MLP) with a final softmax layer for classification. We use binary cross entropy as the loss function.

6.4 Experiments and Results

We experimented with the RTP detection problem that was previously attempted with the MAPLE model in chapter 5 in order to assess the ability to detect weakly signatured data. Then, we extend this to H.225 detection, which is a similarly weakly signatured data transfer protocol. Lastly, the model performance and throughput is assessed for its application to deep packet inspection and deep learning at scale.

All tests were performed on a single CPU of a 1.6 GHz dual-core Intel i5 processor with 16 GB DDR3 RAM. In the binary confusion matrix, true positive (TP) indicates correct classification of data as RTP. True negative (TN) is correct classification of data as non-RTP. false negative (FN) implies incorrect identification of traffic as non-RTP, and false positive (FP) is the incorrect classification of data as RTP. We use measurements of precision, recall, and F1-score for model evaluation, defined as follows:

$$\begin{aligned} Recall = \frac{TP}{TP+FP} \; Precision &= \frac{TP}{TP+FN} \\ F1\ Score &= \frac{2(P \times R)}{P+R} \end{aligned} \tag{6.2}$$

Epochs represent the number of passes made over the training data in order for the neural network to fine tune itself under supervision. Although more epochs does not guarantee better performance, it is observed that a larger number of epochs within reason is a good method to improving metric results. All tests were performed with undersampling to appropriately balance the dataset, and a 60/40 training and test split.

6.4.1 RTP Detection

The primary motivation for designing the density analysis model was to capture the spatial hidden representations which may be present in geometric transformations of packet payloads as dynamic ports and weak signatures make traditional signature-based or port-based methodologies ineffective. The classification results of the Date model are provided in Table 6.2. Table 6.1 show the combined Alpine and Palm technique's results for the RTP classification task. There is clear improvement for the Date model.

Table 6.1: Classification results of RTP vs non-RTP detection for the LSH model.

Class	P	R	F1
Non-RTP	0.73	0.77	0.75
RTP	0.76	0.71	0.73

Table 6.2: Classification results of RTP vs non-RTP detection for the Date model.

Class	P	R	F1
1 Epoch			
Non-RTP	0.85	0.84	0.84
RTP	0.83	0.85	0.84
10 Epochs			
Non-RTP	0.94	0.81	0.87
RTP	0.83	0.95	0.89
20 Epochs			
Non-RTP	0.96	0.79	0.87
RTP	0.82	0.96	0.89

6.4.2 H.225 Detection

A similar detection problem exists for H.225 signaling protocol, which handles the registration and call setup for certain VoIP architectures using the H.323 protocol suite. Like RTP, the H.225 data may arrive to an endpoint first before the H.323 data, and also suffers from a weak signature. Thus, we additionally tested DATE's ability to detect H.225 traffic from non-H.225 traffic. Our setup of the dataset labels and the DBSCAN configuration was similar to the RTP test and results are given in Table 6.4. The performance of the LSH models is provided in Table 6.3 for a baseline.

6.4.3 Model Performance and Throughput

In order to benchmark the system for real deployment, we recorded the time classification took. While this is heavily system dependent and showed some variance

Table 6.3: Classification results of H.225 vs non-H.225 detection for the LSH model.

Class	P	R	F1
Non-H.225	0.62	0.64	0.63
H.225	0.63	0.61	0.62

Table 6.4: Classification results of H.225 vs non-H.225 detection for the DATE model.

Class	P	R	F1
1 Epoch			
Non-H.225	0.98	0.83	0.90
H.225	0.85	0.99	0.92
10 Epochs			
Non-H.225	0.91	0.87	0.89
H.225	0.88	0.92	0.90
20 Epochs			
Non-H.225	0.98	0.83	0.90
H.225	0.85	0.99	0.92

across parameter tuning or model configuration, the average classification time for a single packet by DATE was 0.15823 seconds.

6.5 Discussion and Limitations

Of the models tested so far in this work, DATE is the most computationally complex with the lowest throughput. In future work, we propose implementing multi-threading and multi-core processing as potential optimizations. We observed that point cloud generation was a pain point for the system in terms of cycles, and propose offloading such repetitive calculations to a specialized hardware such as FPGA [111] when available in the deployed system. Because DATE is able to normalize data and map it into a three-dimensional cluster space, it would be well-suited to heterogeneous data such as sensor data combined with packet data from an internet of things device. This would create a multi-embedding capable of correlating these data inputs for machine learning tasks. Simpler problems may be solved with more naÃ¯ve approaches such as regular expression matching on plaintext SIP traffic, or using the locality-sensitive hashing strategy toward device fingerprinting.

We propose both Maple and Date as potential methods for generating hidden, latent-space representations of traffic as their capabilities extend to different problem areas. Maple works well on input data such as a payload which may be distinct one data sample from another, but can be divided and matricized into units for comparison (i.e. turned into a grayscale image of uniform dimension). On the other hand, Date has the potential to expand to include other non-network features and represent more heterogenous data. For example, input sensor data from IoT devices or light detection and ranging (LiDAR) equipment have used DBSCAN for data processing and normalization [112]; it could be combined with cloud/network data inputs as an additional embedding model for classification problems. While for the RTP problem Maple provides high accuracy with more efficiency than Date, there is a trade-off within the embedding space as Date may be able to represent data of higher complexity in other problems for future work.

CHAPTER 7: FORAGER - TOWARDS MULTI-EMBEDDINGS AND ENSEMBLE CLASSIFICATION

7.1 Introduction

In the previous work with ALPINE and PALM we hypothesized that enabling both models to run together and combine their votes would yield the most accurate and agreed-upon classification results. Multiple classifier systems (MCS) are ensembles and are widely used in pattern recognition applications and recognized as more effective than any single embedding or classifier. There are at least three reasons identified by Dietterich as to why multi-classifier approaches perform well [113]:

- Statistically, there is a set of H hypotheses for which the learning algorithm is performing a search for the best hypothesis. The risk of choosing the wrong classifier is reduced when using multiple for unseen data.

- Computationally, optimal training is an NP-hard problem. Because the optimum depends on the starting point, the optimal learning algorithm may be different. By using an ensemble, a better approximation of the unknown function may be obtained.

- Representationally, the weighted sums of hypotheses from H allows for the expanding of space of representable functions. Simply put, the more embeddings we have, the more features, and the more ways to represent and compare data.

In addition to multiple classifiers, in the workflow process using multiple types of embeddings has proven more effective in several recent, real systems than any single approach [?, ?, ?, ?, ?, ?, ?, ?, ?].

As we have seen in previous chapters, each model unveils different hidden representations which are appropriate for different classification problems and contexts. In the real network environment, these classification problems are not isolated. Traffic may need to be distinguished between encrypted and plaintext, for example, and then protocols identified. Then, encrypted traffic may be profiled, or the plaintext traffic characterized by embedding for user fingerprints. Thus, a combined approach or toolkit is apt for the many kinds of problems real world network analysts face.

To meet this need, we propose FORAGER, a combination of data mining and hidden representation learning approaches to deep packet inspection and network traffic classification. This highly configurable toolkit includes the ability to extract and transform header features into locality-sensitive hashes indexed in an LSHForest using the ALPINE technique. Payloads may be combinatorally analyzed using either the PALM, MAPLE, or DATE approaches, or a combination of votes from these models. By considering the data from multiple angles, we achieve results of higher accuracy and a much greater understanding of the network traffic than other state-of-the-art systems.

To illustrate the capability of FORAGER and its diversity of traffic applications, we performed a case study on profiling the traffic on port 443. Due to the rise of encryption and its standard usage for Hypertext Transfer Protocol Secure (HTTPS) encrypted traffic, port 443 is often left un-monitored by packet inspection on large-scale surveillance systems or default-allowed by firewall software. While the majority of internet traffic is now safely secured behind transport layer security, there are threat actors who have found ways to utilize port 443 as a covert data channel for surveillance and firewall evasion, malware transport, data exfiltration, and nefarious activity under the guise of internet anonymity. For increased security and surveillance, it is necessary to be able to filter legitimate, HTTPS traffic from illicit activity on this port. We propose using FORAGER, our network traffic classification and profiling toolkit, to dissect this traffic stream for hidden content. Using FORAGER and its individual models, we are able to extract plaintext and compressed traffic from encrypted streams with high accuracy, profile Tor-based, VPN-based, and encrypted traffic streams by application and data type, and identify malicious instances of particular protocols designated to run over HTTPS. Our models combine several approaches to hidden feature extraction from packets via spatial representation learning to precisely and efficiently classify packets, fortifying network security while still preserving user privacy through this crucial gateway.

7.2 Previous Work

In our threat model and in the works we compare against [9, 8, 11], we choose to prioritize making classification using only a single packet from any point in the overall traffic flow. This contrasts systems which require entire flows in order to classify [10, 11, 12, 13, 14].

7.2.1 VPN Traffic Profiling

Deep Packet [11] uses a CNN to perform traffic classification into traffic types and application types as labeled in the VPN/non-VPN UNB ISCX dataset [85]. Their solution achieves 94% accuracy classifying traffic by type, and 97% accuracy in classifying by application. Like our work, they use a single packet in the classification task, and do not rely on flow-based features. We note that their work does not perform as well in Tor traffic classification, leading us to conclude that Deep Packet may not generalize as well as FORAGER. Cui et al use only header information from the ISCX datasets to classify traffic over flows with 99% accuracy [19]. While a novel solution with enhanced privacy as it considers only header features, it still requires flow-based features of the header and therefore multiple packets. Zou et al use the spatial features of the first packet of a flow extracted and applied to a CNN and analyze the time series features of the next three packets using a long short-term memory network (LSTM) [20]. This method shows high accuracy on the dataset but requires flow-based information.

7.2.2 Darknet Detection and Profiling

Lotfollahi et al also study the ISCX dataset for Tor data and profile again by application and traffic type using a single packet approach [11]. Tor traffic profiling has been further explored by Lashkari et al [8] in their DIDarknet system using a single packet, where they also proved accurate results using the ISCX Tor dataset. Other works have used flow-based features for both detecting Darknet activity (binary Tor/non-Tor classification) and profiling Tor traffic into specific applications or traffic types (e.g. streaming, chat, web browsing). Iliadis et al [114] use an amalgamated version of the ISCX Tor and VPN datasets, re-labeled CICDarknet2020, in order to create a binary classification of Darknet and regular traffic, and a multi-class problem of Tor, non-Tor, VPN, and non-VPN. We perform a similar analysis in our experiments and results of the same data. The work uses all 85 features, including flow-based features, and the full traffic information. Using a set of machine learning algorithms, they are able to achieve up to 98% accuracy; however, in comparison our system uses only a single packet and a minimal set of features to achieve similar results. Ma et al [115] also achieved noted success in the community classifying the same dataset using a CNN with Root Mean Square (RMS) propagation. They also require flow-based features of the dataset, and would therefore require multiple packets and flow information in a real system. They also only consider Darknet detection without additional profiling and achieve 95% accuracy, lower than ours in the same task. Sarkar et al [116] introduce a Generative Adversarial Network (GAN) in order to stabilize their deep learning model across the same ISCX Tor dataset, and achieve approximately 98% accuracy, as well. Using flow-based features, they are able to provide a 95% accuracy in the profiling task across the eight described traffic classes in the dataset; however, they do not provide the results broken down individually so it is difficult to conclude if the model performs evenly well across classes or not. Choorod et al [9] use a statistical analysis solely of the extracted features of the encrypted packet payload. They find that certain sizes are common across traffic types, as well as certain regions of the encrypted payload. They use Charcount from the Posit Text Profiling Toolset [117] as a ratio value for input to their decision tree-based algorithm. They achieve highly accurate results we compare against in the Experiments section of this work.

7.2.3 Filtering Compressed/Plaintext Traffic

Plaintext filtering can be done through statistical entropy tests [118] and has been used to find private medical or geo-location data [118, 119], covert malware [120], and general unencrypted traffic like file transfers or emails. Modern compression and encryption algorithms present such similar high entropy that distinguishing between them is a difficult problem. The HEDGE [121] system is able to parse encrypted from compressed traffic with between 60-94% accuracy using the Chi Square and a subset of the NIST SP 800-22 tests dependent upon the size of the packet used. They compare against Hahn et al, who achieved a maximum of 66% accuracy using machine learning models [122].

7.2.4 Intrusion Detection over HTTPS

Zain ul Abideen et al [123] consider the problem of detecting and profiling VPN traffic on port 443, where their system uses non-encrypted, packet level features to determine several VPN types in the traffic flow. They distinguish between VPN configurations, including Tor. Furthermore, they employ a technique using DNS queries to identify malicious or illegitimate VPN servers. This is an interesting notion which may be challenged by the increased use of encrypted DNS and techniques like DNS-over-HTTPS. Identification of SSH flows in encrypted traffic has been accomplished with success when using flow data [124]. Detecting brute-force and dictionary attacks over SSH is also a research problem area from which our work on port 443 is inspired [125].

7.3 Methodology

7.3.1 Threat Models

To bring the problem to life, we envision a criminal case in our threat model. Bob is a member of an organization embezzling funds through an international shipping company. There are several nefarious tasks which Bob needs to accomplish that he could utilize port 443 exploitations to achieve:

- Create a ***covert data channel*** in order to pass along sensitive information like receipts and records while avoiding detection,
- ***Communicate anonymously*** with criminal network colleagues to avoid tracing of colleagues, his own geolocation data, and evidence,
- Establish ***remote access*** to a machine from endpoints configured to disallow ports like 22,
- ***Exfiltrate data*** from a secure source.

Alice is an intelligence officer who is monitoring Bob's network data with reasonable suspicion that he is engaged in criminal acts. We will discuss vulnerabilities in Alice's current surveillance framework and how to use FORAGER to capture and record evidence which can be used to thwart Bob's plans and provide legal support in the case against his embezzlement and fraudulent company.

7.3.1.1 Real-Time Monitoring

In our scenario, Alice possesses a middlebox technology configured to monitor traffic on port 443. It is common for routers or middleboxes to default-ignore traffic on port 443 as there is usually an overwhelming amount of encrypted traffic on this port that is unprocessable [126]. Attackers who are aware of this strategy may send unencrypted or sensitive information over port 443 in order to avoid surveillance and detection. In criminal operations, these packets could contain significant data. Thus, it is important for middlebox technologies to have a methodology for scanning traffic on port 443 to determine if the data is truly encrypted. If the data is plaintext, it may be directly processed. If it is highly entropic but only compressed, follow-on processing may be capable of decompressing the data. Currently, Alice configures her middlebox to default-ignore traffic on port 443, instead only monitoring port 143 for unencrypted emails. In order to avoid detection by possible surveillance technology, Bob routes his emails through port 443. The receiver is aware of this and listens for this traffic with the IMAP server on the configured port so that it is received. Thus, Bob is able to transmit content in a way that is receivable by the partner as email/IMAP at no additional security overhead but is undetected as Alice's middlebox does not scan port 443.

7.3.1.2 Tunnel Tracking

Tor allows anonymous internet space for pervasive action and may be used to access content which is blocked by firewalls, whether it be censored or illegal activity. While possibly encrypted and encapsulated with layers of addressing in order to anonymize routing, traffic may be synchronized by timing and further analysis in order to determine endpoints [127]. Thus, tracking Tor and being able to further profile Tor usage and content is a useful operation for intelligence and law enforcement. Network Address Translation (NAT) in VPNs prevents traditional ISP tracking. The ability to detect VPN traffic streams and profile their traffic and applications can be additionally useful for network monitoring. In the scenario, Bob is using a VPN to connect to a forum website. He wishes to remain anonymous during this communication. Alice is following Bob's messages, and sees that he regularly communicates using HTTP and IRC protocols on this platform. She configures her middlebox device to promote traffic on several five-tuple-based traffic flows which she knows to be currently associated with Bob's devices. However, dynamic re-configuration of the tunneled network layers causes Alice to lose the data stream and she is no longer able to monitor his messages and record the activity.

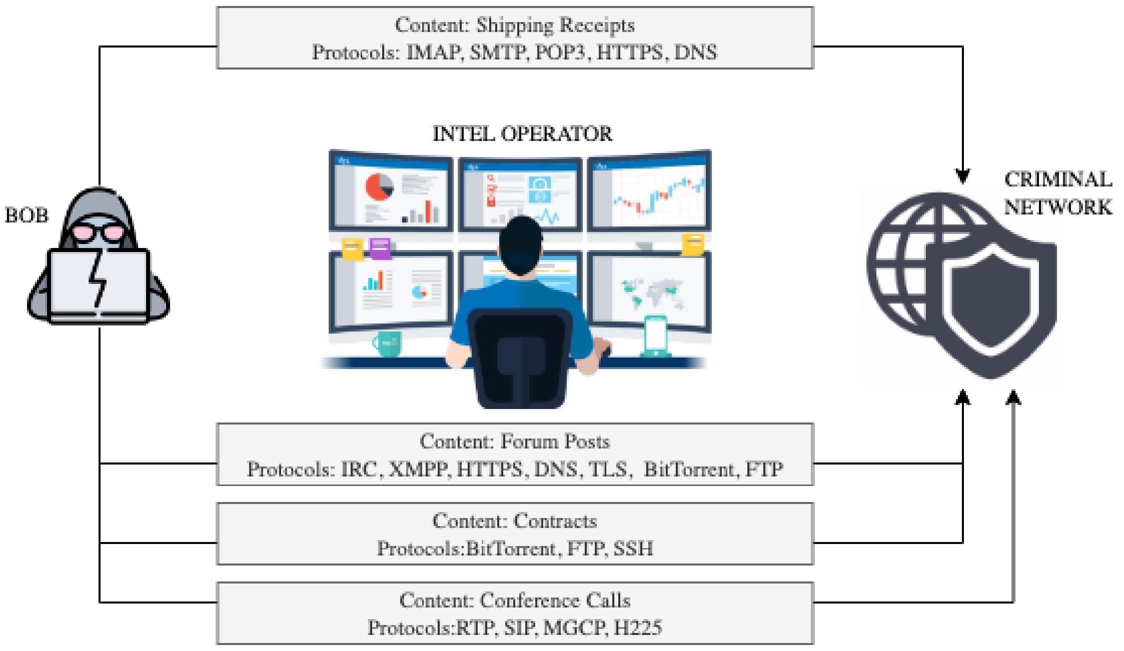

Content: Shipping Receipts
Protocols: IMAP, SMTP, POP3, HTTPS, DNS
INTEL OPERATOR
BOB
CRIMINAL NETWORK
Content: Forum Posts
Protocols: IRC, XMPP, HTTPS, DNS, TLS, BitTorrent, FTP
Content: Contracts
Protocols:BitTorrent, FTP, SSH
Content: Conference Calls
Protocols:RTP, SIP, MGCP, H225

Figure 7.1: A depiction of intelligence operations intercepting various types of traffic Bob is transmitting in connection to his extended criminal network.

7.3.1.3 Covert Data Channels In

Domain name security (DNS) has long been exploited by attackers due to many security loopholes and its historical transmission in plaintext. Malicious actors may embed malware or sensitive data into DNS records [128]. This becomes even harder to detect when the tunnel is then encrypted, as is the case with DNS over HTTPS [129]. Since its inception, DNS over HTTPS has been adopted by mainstream browsers like Firefox and Chrome to prevent man-in-the-middle attacks and snooping on DNS traffic. While enhancing user privacy, this has contraversely become a field of opportunity for cyber attackers as DNS traffic that is now encrypted may bypass traditional DNS security systems. Thus, the DNS-over-HTTPS (DoH) pipeline can become both a covert channel for malware entering systems as well as data being exfiltrated out. For example, in 2020 the Iranian threat actor group APT34 used DNSExfiltrator2, an open-source DoH tool, to laterally move data to their networks [130]. Malware such as Godlua [131] and PsiXBot [132] have also been spotted in the wild along DoH channels. As an example in our scenario, imagine Bob wants to track activity or exfiltrate data from a server in a competitor's private network. In order to record this information, he needs to plant a trojan virus on this server. Using DNS-over-HTTPS as a covert data channel, Bob hosts the trojan on a website and uses social engineering to engage a user to click a link, querying the domain and returning the data which contains the trojan. This malware remains undetected as port 443 traffic is not scanned.

7.3.1.4 Covert Data Channels Out

Many enterprise network environments restrict outbound traffic to ports 80 and 443. Users may re-reroute DNS from the standard port 53 to avoid defeat by firewalls, ISPs, or government or enterprise networks [126]. Similarly, firewalls can also be bypassed using SSH over port 443. SSH, or secure shell protocol, gives users a method of accessing another computer over an insecure network. While typically transmitted over port 22, many firewalls default to defeating traffic over this port. Instead, they allow traffic over port 443. Packages exist such as `stunnel` which make SSH tunneling over 443 simple for any internet user [133]. Creating a covert data channel over SSH can also be a gateway for cache attacks as it is necessary to transmit or exfiltrate the data when accessed [134]. In our scenario, Alice's middlebox technology is monitoring traffic on port 22 (SSH) for secure shell sessions. Bob needs to SSH into a machine and copy over some documents. Instead of relying on traditional channels, Bob has re-routed his SSH connection through port 443, creating a covert data channel. Thus, Alice is missing the mission critical data of the file transfer.

7.3.2 System Design

The FORAGER system is a highly adaptable traffic profiling multi-tool capable of representing packets in multi-dimensional space through configurable modules. Users can enable up to four different methods of latent representation for the packets. Each of these modules assesses different portions of the packet data with various hidden representation learning approaches. Specifically, ALPINE and PALM [135] generate one-dimensional locality sensitive hashes which represents single sets of features from the header and payload. MAPLE and DATE generate two-dimensional images and three-dimensional point clouds with clusters, respectively. Votes are provided to the overall framework and used in lookup to generate a classification report. The whole system is depicted in Figure 7.2.

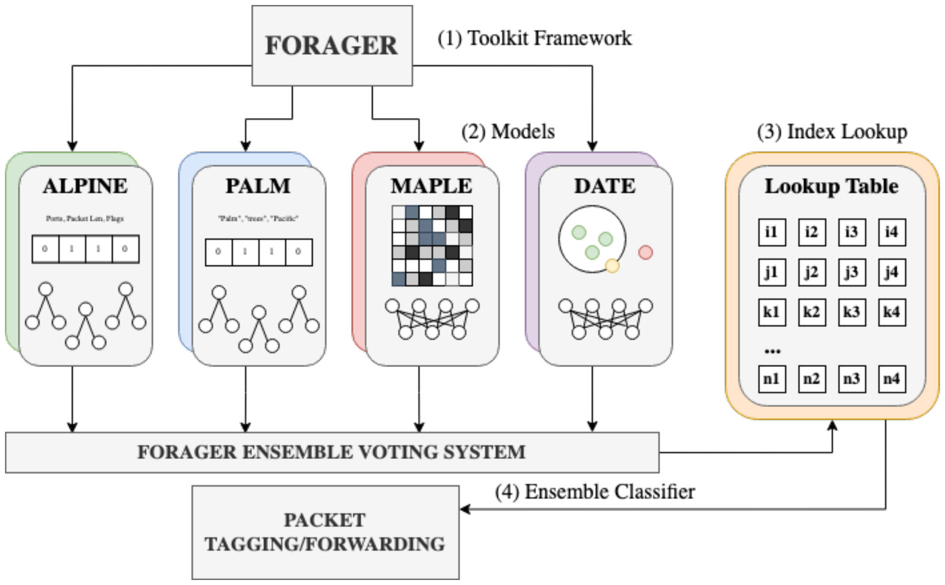


Figure 7.2: System diagram of FORAGER.

In the training phase, data is passed in with a label and indexed respective to class. This index is stored in a lookup table inside FORAGER. Models are enabled or disabled and given a number of votes m per model. This allows the user to more heavily weight one model's opinion over another, further contributing to the flexibility in real deployments. A few additional hyperparameters may be configured for the neural network models such as epochs for training, and data is processed and weights may be saved for later use. In the testing phase, test data is processed in a similar manner and the input passed through the classifier. Forager amalgamates the votes across models and uses its lookup table to return a classification result. In a real packet processing pipeline, we could add this result as part of metadata or an encapsulation header for the packet for downstream processing. The current implementation generates a report of classification results which may be returned to an administrator for further review and possible identification of traffic on non-standard or unexpected ports which could be malicious.

7.3.3 Datasets

To create a pipeline of data representative of a diverse capture on port 443, we utilized several public datasets created by and widely used in the research community for network traffic classification.

7.3.3.1 ISCX 2016 VPN-nonVPN Dataset

We used the UNB ISCX 2016 VPN-nonVPN dataset [85]. This dataset is a real-world simulation capture created by the Canadian Institute for Cybersecurity (CIC) specifically for encrypted VPN traffic classification [**?**]. The traffic was captured using Wireshark and tcpdump and generated using OpenVPN; it contains 28GB of traffic. The packets are a mixture of encrypted and compressed or plaintext traffic. Because these are real-world captures, there are some packets such as ICMP and ARP which are service-level and irrelevant for application and traffic type classification. For this reason, we discard non-TCP/UDP packets as part of the process. Previous work [11, 136, 43] in machine learning-based traffic classification has made use of this public dataset and made similar modifications. A breakdown of the dataset is provided in Table 7.1. The data is divided into the following categories:

- *Traffic Type* - There are seven types of traffic: Browsing, File Transfer, Email, Chat, Streaming, VoIP, and P2P.

- *Application* - Several of the PCAPs were associated specifically with one application such as Skype or Google Hangouts. The applications we uniquely identify are Pidgin (IAM and ICQ chat), BitTorrent (P2P), Facebook, Skype, Google Hangouts, Spotify, Vimeo, Youtube, Netflix, Thunderbird (Email), and Filezilla (FTP).

Table 7.1: ISCX VPN/non-VPN and Tor/non-Tor Datasets

Traffic	**Content**
Browsing	Firefox and Chrome
Email	SMTPS, POP3S and IMAPS
Chat	ICQ, AIM, Skype, Facebook, Hangouts
Streaming	Vimeo, Youtube, Spotify
File Transfer	Skype, FTPS and SFTP using Filezilla
VoIP	Facebook, Skype, Hangouts voice calls
P2P	uTorrent, BitTorrent, Vuze

Application	**Benign**	**VPN**	**Tor**
AIM chat	6K	1.4K	2K
Email	50K	25K	163K
Facebook	1837K	36K	88K
Filezilla (FTPS)	312K	315K	143K
Filezilla (SFTP)	629K	182K	176K
Gmail	15K	–	–
Chrome	–	–	356K
Hangouts	1871K	151K	1114K
Pidgin (ICQ)	4K	7K	2K
Netflix	455K	1433K	–
SCP	777K	–	–
Skype	2261K	1069K	564K
Spotify	63K	193K	118K
Bittorrent	–	133K	–
VoipBuster	1017K	823K	–
Vimeo	214K	591K	288K
Vuze	–	–	264K
Youtube	377K	333K	261K

7.3.3.2 Tor Datasets

The Canadian Institute for Cybersecurity similarly created the UNB ISCX 2016 Tor-nonTor [86] dataset which captures data routed over Tor. The traffic was generated using Whonix [1] and captured using Wireshark and tcpdump; the Tor dataset contains 22GB of traffic. The benign traffic used for comparison in their traffic profiling work [18] is the same from the VPN dataset, establishing a good baseline for our work in combining these two. A gateway was configured to convert non-encapsulated traffic into Tor PCAPs which were then routed to the Tor network. Because Tor is a circuit-oriented network, all traffic from the gateway to the entry node will be routed through the same connection [18] which could cause over-fitting to a particular dataset. In order to better simulate the diversity of network routing methods, we added as another Tor sample PCAPs from Skynet, a Tor-powered botnet with publicly available datasets for research purposes [84]. For a botnet, the advantage of Tor is hidden services. There is no way to trace the original IP address of a hidden server which is published with its .onion pseudo-domain. All botnet traffic is encrypted. The malware which infects the host opens a SOCKS proxy on port 55080 reachable through the .onion domain, and can then run a number of bundled attacks and operations like bitcoin mining, data exfiltration, and DDoS attacks [84]. Thus, we leverage evidence of this botnet implementation as an additional Tor data connection.

7.3.3.3 CIRA-CIC-DoHBrw-2020 Dataset

CIC has also made the CIRA-CIC-DoHBrw-2020 dataset containing DNS-over-HTTPS (DoH) and non-DoH traffic on port 443, with a secondary layer of benign versus malicious DoH traffic [128]. The traffic was generated by querying the top 10K Alexa websites. For non-DoH traffic, an HTTPS request is sent for the website with regular, plaintext DNS. Firefox and Chrome browsers with the configured DNS-over-

[1]https://www.whonix.org

HTTPS setting were used for benign DoH traffic. For each of the browsers, there were four DNS services used: Cloudflare, AdGuard, Google, and Quad9. Finally, for malicious DoH traffic, tunneling tools dns2tcp, DNSCat2 and Iodine were used to created encrypted TCP covert data channels for transmission. A domain and authoritative name server were established and the DoH tunneling tool used in the client/server interaction. The packet pre-processing we performed on this data was similar to the VPN and Tor dataset processing; a distribution of the data is provided in Table 7.2.

7.3.3.4 EMews SoH Dataset

SSH is another exploitable protocol for both malware injection and data exfiltration over HTTPS. In 2018, Ricks et al [137] created an SSH over HTTPS (SoH) simulation using the eMews framework and CORE to generate packet traces. The network consists of 1022 nodes, 36 of which incorporate the SSH sessions and the others perform various web-crawling activity for benign comparison. EMews [2] itself is an open-source, large-scale network traffic data generation tool designed to emulate networks employing protocols which require human interaction (ex: starting an SSH connection). Sample counts for this dataset are provided in Table 7.3.

Table 7.2: Summary of CIRA-CIC-DoHBrw-2020 Dataset

Classification	**Application**	**Sample Count**
non-DoH	Chrome	542K
	Firefox	356K
Benign DoH	Chrome	3.5K
	Firefox	16.2K
Malicious DoH	dns2tcp	168K
	DNSCat2	36K
	Iodine	47K

[2]https://mews.sv.cmu.edu/research/emews/

Table 7.3: Summary of eMews SSH over HTTPS Dataset

Classification	**Sample Count**
HTTPS	542K
SSH over HTTPS	3.5K

7.4 Experiments and Results

We use measures of precision, recall, and F1 score for classifier assessment. In the binary confusion matrix, true positive (TP) indicates correct classification of data as some protocol class C. True negative (TN) is correct classification of data as *not* class C. false negative (FN) implies incorrect identification of traffic as not C, and false positive (FP) is the incorrect classification of data as C.

$$R(C) = \frac{TP}{TP + FP},\ P(C) = \frac{TP}{TP + FN},\ F1(C) = \frac{2(P \times R)}{P + R} \tag{7.1}$$

A high recall indicates that in the binary problem, much of the traffic is correctly identified as the positive class. High precision in the model shows the quality of positive classifications, i.e. indicates how many were correctly classified out of everything said to be in the positive class. A model with low recall and high precision will likely mis-classify traffic which does belong in the positive class (high rate of false negatives), whereas a model with high recall and low precision will produce more false positives. In the following scenarios, we analyzed different potential traffic classification problems for port 443 based on the described attacks or opportunities from the threat models.

7.4.1 VPN Traffic Profiling

We analyzed the ISCX VPN/non-VPN dataset according to the applications and traffic types identified by the researchers who generated the data and Lotfollahi et al [11] who used the same data for per-packet classification in their Deep Packet models. They used a convolutional neural network and stacked autencoder model for implementations. We used three different configurations of Forager for the traffic and application classification tasks. For each problem, we ran a combination of ALPINE and one of the payload transformation models. For application classification, DeepPacket's CNN model achieved very high F1 scores for most of the applications. Overall, each model is quite successful in distinguishing applications without inspecting packet contents, preserving user privacy while still accomplishing the mission. In the threat model, Alice could use our tool to better understand what kind of internet behavior Bob is engaging in over his VPN connection. We notice in both application and traffic classification tasks, our FORAGER models achieve some improved results in the VoIP categories. Our models also perform significantly better in profiling the benign/non-VPN traffic types, up to 0.13 higher than the stacked auto encoder in the VoIP category as well. When ICQ and AIM were combined under the Chat category, FORAGER achieves much higher results than any other model. For all models, distinguishing ICQ and AIM traffic was a difficult task. In further investigation into the network environment, we found both of these traffic streams were generated from the Pidgin messaging application, which makes sense that they were misclassified as one another in all the models. We also note that apps which use similar protocol stacks or likely similar APIs tend to be misclassified as one another. for example, FTPS and SFTP (both using FTP) are more difficult to distinguish from one another than either against SCP using SSH [11]. This is an important note for training and defining classes for neural network models in general, as well-defined classes or clusters make classification a clearer task. One significant difference in the experimental setup of Deep Packet versus FORAGER is that DeepPacket's CNN requires 300 epochs to train the entire network. MAPLE achieves near comparable results at a mere 10 epochs of training on the same data and split. DATE also achieves good results only training with 10 epochs on its statistical model. We ran additional tests, increasing the number of epochs in order to determine if this improved our models' efficacy.

In each case, 10 epochs proved to be sufficient for training FORAGER to maximum performance capability.

7.4.2 Tor Traffic Profiling

Another type of tunneled traffic we want to identify and profile on port 443 is anything running over Tor. We again ran our three configurations of the FORAGER toolkit against the Tor dataset. Deep Packet [11] also ran their SAE and CNN models against this data, but only report a portion of results. We provide a comparison against what they do report for completeness, and note that the FORAGER models are a dramatic improvement in terms of capability. Their model is unable to handle most of the application classes, whereas FORAGER can adapt to all applications and returns results as high as those in the VPN classification task. This result demonstrates the versatility of FORAGER compared to the state-of-the-art and current existing frameworks. Table 7.5 shows results of the application classification task. We also show comparative results of traffic type classification with work performed by Lashkari et al [8] in their DIDarknet system which uses a two-dimensional convolutional neural network to characterize VPN and Tor traffic. It is important to note that unlike Deep Packet and FORAGER models, DIDarknet's CNN relies on flow-based features. Both we and Deep Packet prefer a stateless approach for best-effort, per-packet classification and use features of the individual packets themselves. Choorod et al [9] provide results of single packet classification of the Tor dataset using features extracted with Charcount from the Posit toolkit along with a decision tree-based algorithm called J48. This performed remarkably well on the traffic type classification task, and we also compare against their results. They marginally outperform the FORAGER system; but both identified Tor traffic with a much greater accuracy and F1 score than the flow-based system. Choorod et al did not provide results for application-based identification, and it is unclear if their method would generalize to any other problems or has been attempted on any other data sets. The experiment results demonstrate

Table 7.4: Results for classifying VPN data.

Application	DeepPacket-CNN			DeepPacket-SAE			ALPINE+PALM			ALPINE+MAPLE			ALPINE+DATE		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
AIM chat	0.87	0.76	**0.81**	0.76	0.64	0.70	0.80	0.55	0.65	0.80	0.57	0.67	0.67	0.69	0.68
Email	0.97	0.82	0.89	0.94	0.99	**0.97**	0.81	0.76	0.79	0.74	0.80	0.77	0.86	0.73	0.79
Facebook	0.96	0.95	**0.96**	0.94	0.95	0.95	0.96	0.90	0.93	0.97	0.91	0.94	0.98	0.90	0.94
FTPS	1.00	1.00	**1.00**	0.97	0.77	0.86	1.00	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Gmail	0.97	0.95	**0.96**	0.93	0.94	0.94	0.96	0.87	0.91	0.94	0.90	0.92	0.95	0.90	0.92
Hangouts	0.96	0.98	**0.97**	0.94	0.99	0.97	0.98	0.89	0.93	0.96	0.91	0.94	0.96	0.91	0.94
ICQ	0.72	0.80	**0.76**	0.69	0.69	0.69	0.47	0.93	0.63	0.59	0.89	0.71	0.60	0.89	0.72
Netflix	1.00	1.00	**1.00**	1.00	1.00	1.00	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99
SCP	0.97	0.99	0.98	1.00	1.00	**1.00**	0.98	0.91	0.94	0.98	0.92	0.95	0.97	0.91	0.94
SFTP	1.00	1.00	**1.00**	0.70	0.96	0.81	0.99	1.00	1.00	0.99	1.00	0.99	0.99	1.00	1.00
Skype	0.94	0.99	0.97	0.95	0.93	0.94	0.99	0.95	**0.97**	0.98	0.92	0.95	0.98	0.92	0.95
Spotify	0.98	0.98	**0.98**	0.98	0.98	0.98	0.99	0.92	0.95	0.99	0.95	0.97	0.99	0.95	0.97
Torrent	1.00	1.00	**1.00**	1.00	1.00	1.00	0.99	0.97	0.98	0.98	0.99	0.98	0.98	0.99	0.98
VoipBuster	0.99	1.00	0.99	0.99	0.99	0.99	1.00	0.98	0.99	1.00	0.98	0.99	1.00	0.99	**0.99**
Vimeo	0.99	0.99	**0.99**	0.99	0.98	0.98	0.99	0.98	0.99	0.97	0.98	0.99	0.98	0.98	0.98
Youtube	0.99	0.99	**0.99**	0.98	0.99	0.99	0.99	0.94	0.96	0.96	0.96	0.96	0.96	0.96	0.96
Traffic Type	DeepPacket-CNN			DeepPacket-SAE			ALPINE+PALM			ALPINE+MAPLE			ALPINE+DATE		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
Chat	0.84	0.71	0.77	0.82	0.68	0.74	0.95	0.81	**0.87**	0.93	0.64	0.76	0.82	0.72	0.77
Email	0.96	0.87	0.91	0.97	0.93	**0.95**	0.76	0.99	0.86	0.71	0.74	0.73	0.75	0.67	0.71
File Transfer	0.98	1.00	**0.99**	0.98	0.99	**0.99**	0.99	0.91	0.95	0.74	0.93	0.82	0.74	0.93	0.82
Stream	0.92	0.87	0.90	0.82	0.84	0.83	0.99	0.98	**0.99**	0.99	0.98	**0.99**	0.99	0.98	**0.99**
VoIP	0.63	0.88	0.74	0.64	0.90	0.75	0.99	0.95	**0.97**	0.99	0.94	0.96	0.99	0.95	**0.97**
VPN: Chat	0.98	0.98	**0.98**	0.95	0.94	0.94	0.96	0.95	0.96	0.96	0.98	0.97	0.96	0.97	0.97
VPN: Transfer	0.99	0.99	**0.99**	0.98	0.95	0.97	1.00	0.98	**0.99**	1.00	0.98	**0.99**	1.00	0.98	**0.99**
VPN: Email	0.99	0.98	**0.99**	0.97	0.93	0.95	1.00	0.98	**0.99**	0.99	0.98	**0.99**	0.97	0.99	0.98
VPN: Stream	1.00	1.00	**1.00**	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.80	0.89
VPN: Torrent	1.00	1.00	**1.00**	0.99	0.97	0.98	0.99	0.99	0.99	0.98	0.99	0.99	0.83	1.00	0.91
VPN: VoIP	0.99	1.00	**1.00**	0.99	1.00	0.99	1.00	0.98	0.99	0.99	0.98	0.99	0.99	0.98	0.99

Table 7.5: Results for classifying Tor data by application.

Application	DeepPacket-CNN			DeepPacket-SAE			ALPINE+PALM			ALPINE+MAPLE			ALPINE+DATE		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
AIM chat	–	–	–	–	–	–	0.86	0.53	**0.66**	0.95	0.50	0.65	0.96	0.49	0.65
Chrome	0.00	0.00	0.00	0.03	0.44	0.06	0.93	0.68	0.78	0.91	0.74	0.81	0.91	0.75	**0.82**
Email	–	–	–	–	–	–	0.94	0.84	0.89	0.94	0.85	0.89	0.94	0.85	**0.90**
Facebook	0.10	0.24	0.14	0.06	0.28	0.09	0.97	0.88	0.92	0.97	0.89	**0.93**	0.97	0.89	**0.93**
FTPS	–	–	–	–	–	–	0.95	0.99	**0.97**	0.95	0.99	**0.97**	0.95	0.99	**0.97**
Gmail	0.97	0.95	0.96	0.93	0.94	0.94	0.94	0.85	0.89	0.89	0.90	0.89	0.94	0.86	**0.90**
Hangouts	–	–	–	–	–	–	0.91	0.93	**0.95**	0.95	0.92	0.93	0.93	0.94	0.94
ICQ	–	–	–	–	–	–	0.40	0.94	0.56	0.51	0.97	**0.67**	0.48	0.98	0.65
Netflix	–	–	–	–	–	–	0.98	0.98	0.98	0.98	0.99	0.98	0.99	0.98	**0.99**
SCP	0.97	0.99	0.98	1.00	1.00	1.00	0.99	0.90	0.94	0.96	0.92	0.93	0.96	0.92	**0.94**
SFTP	–	–	–	–	–	–	1.00	0.94	**0.97**	0.99	0.94	0.97	0.99	0.94	0.97
Skype	–	–	–	–	–	–	0.97	0.91	**0.94**	0.95	0.92	0.93	0.96	0.91	0.93
Spotify	–	–	–	–	–	–	0.79	0.91	**0.85**	0.75	0.95	0.84	0.74	0.95	0.83
VoipBuster	–	–	–	–	–	–	1.00	0.98	**0.99**	1.00	0.98	**0.99**	1.00	0.98	**0.99**
Vimeo	0.44	0.36	0.40	0.05	0.91	0.09	0.98	0.94	0.96	0.95	0.95	0.95	0.97	0.95	**0.96**
Vuze	–	–	–	–	–	–	0.98	0.72	0.83	0.97	0.73	**0.83**	0.97	0.71	0.82
Youtube	–	–	–	–	–	–	0.95	0.94	0.94	0.94	0.95	0.95	0.94	0.96	**0.95**

Table 7.6: Results for classifying Tor data by traffic type.

Traffic Type	DIDarknet			J48			ALPINE+PALM			ALPINE+MAPLE			ALPINE+DATE		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
Audio	0.92	0.92	0.92	0.97	0.97	**0.97**	0.99	0.94	0.96	0.99	0.94	0.96	0.99	0.94	0.96
Browsing	0.55	0.47	0.51	0.99	0.99	**0.99**	0.97	0.94	0.95	0.95	0.97	0.96	0.95	0.97	0.96
Chat	0.90	0.86	0.88	0.93	0.93	**0.93**	0.92	0.74	0.82	0.84	0.80	0.82	0.93	0.70	0.80
Email	0.66	0.67	0.67	0.93	0.93	**0.93**	0.90	0.90	0.90	0.91	0.87	0.89	0.92	0.86	0.89
File Transfer	0.74	0.75	0.75	0.98	0.98	**0.98**	0.99	0.92	0.96	0.99	0.92	0.96	0.99	0.92	0.96
P2P	0.90	0.95	0.93	0.99	0.99	**0.99**	0.96	0.98	0.97	0.95	1.00	0.97	0.96	0.99	0.97
Video	0.82	0.88	0.85	0.98	0.98	**0.98**	0.99	0.97	**0.98**	0.98	0.98	**0.98**	0.98	0.98	**0.98**
VOIP	0.58	0.61	0.59	0.99	0.99	**0.99**	0.72	0.97	0.83	0.82	0.96	0.89	0.71	0.99	0.82

that the system could be used to isolate Tor traffic on port 443, or any traffic stream. In the context of the threat model, Alice can determine if and when Bob is using Tor routing, which is useful for further track and trace. She also has a much greater insight into the traffic type and applications than other models previously gave, advancing

Table 7.7: Results for VPN/Tor classification

Class Set	**Accuracy**	**P**	**R**	**F1**	κ
Tor	0.99	0.99	0.99	0.99	0.99
VPN	0.97	0.98	0.97	0.97	0.98
Standard	0.99	0.99	0.99	0.99	0.99
Class Set	**Accuracy**	**P**	**R**	**F1**	κ
VPN/nonVPN	0.97	0.98	0.97	0.97	0.97
Traffic Type	0.86	0.88	0.86	0.86	0.82
Application	0.92	0.93	0.92	0.92	0.82
Class Set	**Accuracy**	**P**	**R**	**F1**	κ
Tor/nonTor	0.99	0.99	0.99	0.99	0.99
Traffic Type	0.97	0.97	0.97	0.97	0.96
Application	0.89	0.91	0.89	0.89	0.88

the state of the art. A summarization is provided in Table 7.7.

7.4.3 Plaintext Traffic over HTTPS

Our next experiment was to assess the ability of the model to determine if traffic samples on port 443 are truly encrypted packets (TLS) or not. Many firewalls are configured to default-allow port 443 traffic; an attacker may send any content or malware over this port in order to attempt to bypass a firewall. Senders wishing to bypass middlebox technology may send non-TLS data over port 443 to avoid detection. It is thus significant for surveillance and law enforcement operations to monitor traffic on port 443 for non-encrypted data which may be decipherable. IoT devices also can send unencrypted traffic over port 443 [118] which may contain some useful information. Alice finds it incredibly useful to capture all plaintext data where possible as it can contribute to the legal summation of a case. We created two classes of encrypted and non-encrypted traffic, performed random under-sampling, and ran the model, generating results in Figure **??**. The model tends to overselect traffic as encrypted, and underselect traffic which is plaintext. This is not surprising due to the variety of features present in the single plaintext category, versus the lack of common features generally found or uncovered in the encrypted class. In this case, model design may not be well-suited to a binary classification problem because the class *plaintext* is not well-defined. This tuning factor should be considered by users of the FORAGER system during the configuration process.

7.4.4 Compressed Traffic over HTTPS

Compressed versus encrypted traffic both tend to exhibit highly entropic behavior, with the data approaching uniform distribution in more advanced implementations [121]. An attacker may run compressed data over port 443 (or any port) and middleware technology may assume it is encrypted if using statistical tests like Chi-squared or Shannon entropy [138]. Instead, Alice could use our system to isolate compressed traffic from encrypted traffic, and thus be able to decompress and process that information in follow-on applications. In order to run this experiment, we first ran compression algorithms on the payloads of FTP_DATA packets. We chose this traffic type as the payloads are largely text-based and often sent compressed over the wire. We used three standard compression libraries - `gzip`, `zlib`, and `bz2`. The packets were randomly divided into groups, producing 10,519 sample compressed payloads of each type. The encrypted traffic from the ISCX 2016 VPN-nonVPN dataset [**?**] was used for the encrypted traffic type. Figure 7.4 shows FORAGER is capable of identifying compressed from encrypted traffic on port 443 or out of any stream (using ALPINE and MAPLE). Furthermore, we can profile down to specific compression type with great accuracy in Figure 7.5. We can also identify plaintext traffic in Figure 7.3. Alice could use this technology to extract compressed data streams and decompress them for further analysis.

7.4.5 SSH over HTTPS

We want to be able to identify particular instances of certain protocols over an encrypted traffic network; One such protocol of interest is SSH. This is often used to bypass firewall restrictions or create covert data channels. In our experiments,

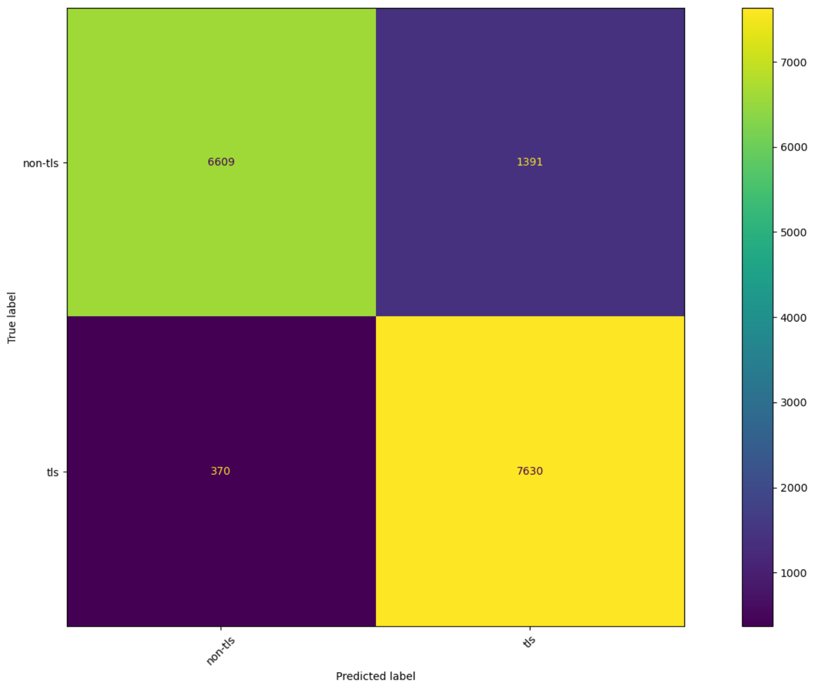


Figure 7.3: Plaintext versus encrypted traffic results.

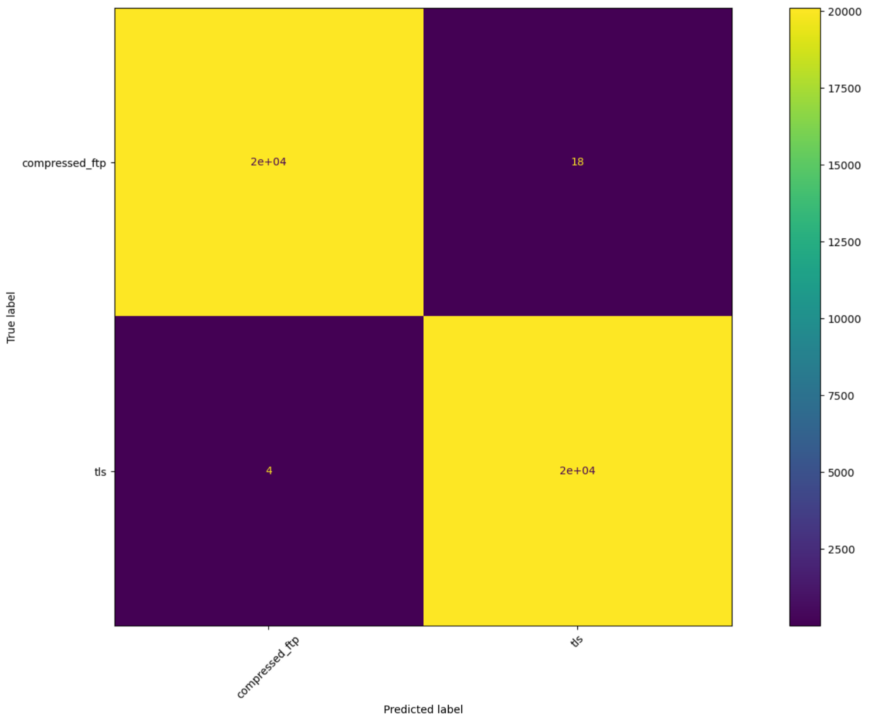


Figure 7.4: Compressed versus encrypted traffic results.

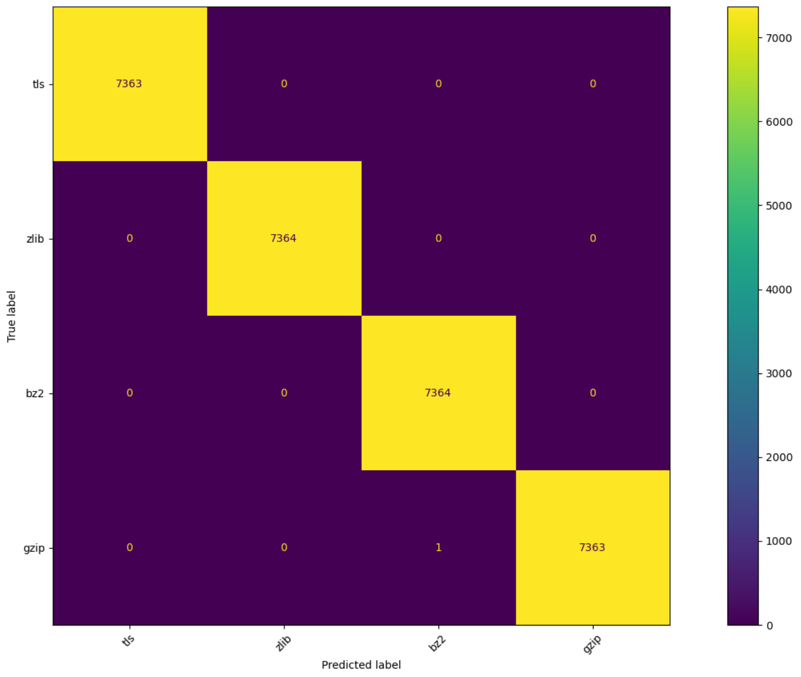


Figure 7.5: Compressed versus encrypted traffic results.

we implemented the same model combinations as with the previous scenarios. The Forager toolkit was highly successful in classifying SSH traffic from the SSH-over-HTTPS dataset generated by the EMews simulator, achieving over 99% accuracy as shown in Table 7.9. Identifying this traffic would allow inspectors to both mitigate covert data channels or monitor them for possible illicit activity. For example, Alice could detect any open covert data channels Bob might attempt to create over SSH.

Table 7.8: Results for encryption and compression detection

Protocol	**P**	**R**	**F1**
Encrypted	1.00	1.00	1.00
Compressed	1.00	1.00	1.00
Encrypted	0.85	0.95	0.90
Plaintext	0.95	0.83	0.88
TLS	1.00	1.00	1.00
BZ2	1.00	1.00	1.00
GZIP	1.00	1.00	1.00
ZLIB	1.00	1.00	1.00

7.4.6 DNS over HTTPS

Our final experiment was to detect DNS traffic running over HTTPS. A perpetual challenge for security research in classification and anomaly detection is the notion of intent; just because something is identified as belonging to a particular class or marked as an outlier does not always correlate that it is malicious. For example, a new instance of a running process can just be that - a user running a new program - and not necessarily an executing malware. Or a malformed data packet may appear different than others in the stream, but that could just be a bad transmission. For DNS over HTTPS, mainstream browsers like Chrome and Firefox employ this technique without malicious intent. As such, being able to determine DNS tunneling as a second layer to the DoH/non-DoH identification is of critical importance. The results in Table 7.9 show that FORAGER rises to meet this challenge, and can distinguish both DoH/non-DoH traffic and then malicious traffic from the identified DoH traffic with high accuracy and few false positives. Thus, any attempt Bob makes at DoH tunneling is now detectable using our toolkit.

Table 7.9: Results for classifying DoH/non-DoH and SoH/non-SoH data.

Model	A+P			A+M			A+D		
Class	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
SSH	1.00	0.99	1.00	0.98	1.00	0.99	1.00	1.00	1.00
Non-SSH	0.99	1.00	1.00	1.00	0.98	0.99	1.00	1.00	1.00
DNS	1.00	0.97	0.98	0.96	0.99	0.98	0.95	0.99	0.97
Non-DNS	0.97	1.00	0.98	0.99	0.96	0.97	0.95	0.97	0.96
Malicous	1.00	0.95	0.97	1.00	0.95	0.97	0.99	0.95	0.97
Benign	0.96	1.00	0.98	0.95	1.00	0.97	0.96	0.99	0.97

7.4.7 Performance Metrics

The experiments were run on a 1.6 GHz Dual-Core Intel Core i5 processor with 16GB of RAM running MacOS Big Sur version 11.6.1. For each of our experiments, we tracked system performance metrics for the models as they performed the classification task and provide this data in Appendix A. This is important for real systems to assess not only the expected tonnage of network traffic they can process, but also to determine what models may be appropriate for deployment. For example, a model whose strategy focuses on online training may not be suitable in an environment who requires high throughput and low downtime, but has a high training time. Or, in some scenarios it may be acceptable to have slightly lower accuracy but a higher throughput rate.

7.5 Discussion and Limitations

Using Forager, we are able to provide new insight into a previously under-analyzed gateway into secure networks - port 443. These techniques enable profiling of encrypted traffic streams while preserving user privacy of the content. We can also use Forager to identify plaintext or compressed traffic from the encrypted streams which may then be forwarded to decompression and DPI tools. Finally, we show the ability of the system to identify not only particular protocols in the encrypted stream, but also distinguish between benign and malicious instances of their utilization. This would be useful, for example, applied in network intrusion detection (NIDS) and prevention systems. For surveillance missions, we show Forager keeps up with the ever-changing, fast-paced real and complex network environment and allows analysts to stay ahead of mission communications and critical data. In seeing through encryption and tunneling, Forager proves to be a versatile solution for real network problems.

For plaintext analysis and keyword searching, using regular expression scanning can still be an effective technique. Automatic generation techniques like RExACtor may also reduce manual overhead required for signature creation and maintenance, and provide useful insight to commonalities across traffic types and protocols. Additional regular expression scanning and generation tools would be a useful addition to the Forager toolkit.

CHAPTER 8: EXPECTED CONTRIBUTIONS AND TIMELINE

8.1 Expected Timeline and Contributions

RExACtor furthers the state-of-the-art in regular expression generation in the following ways:

- adds a component tool for automatically extracting frequent and certain tokens from packet payloads,
- applies genetic sequencing for substring alignment combined with frequent tokens,
- encodes substrings and tokens in an original algorithm for more enriched, expressive regular expressions,
- adds an additional tool for regular expression scanning using state-of-the-art automata software. For

Palm and Alpine provide the following contributions:

- A process for generating multiple locality-sensitive hash embeddings from packets which is highly accurate for identification of many classes, including protocol type, traffic type, application, and more,
- A generalizable framework which applies to many network traffic classification problems, and whose model can be quickly trained and adapted to suit new problems,
- An alternative to regular expression scanning for DPI which scales sublinearly and requires a linear amount of storage space,

- A diverse and unique application of the traffic classification problem with experiments classifying multiple classes of protocols across many traffic types,
- and public datasets and source code for experimental reproducibility.

MAPLE and DATE additionally contribute:

- An algorithm for encoding packet payloads to image-based embeddings for latent representation,
- an empirical evaluation and comparison of CNN models on RTP data,
- an algorithm for generating three-dimensional point clouds from packet data, creating spatial latent representations as a novel method of packet analysis,
- a novel application of density-based cluster analysis on packet payloads.

CHAPTER 9: CONCLUSIONS

Lots of interesting conclusions will be put here.

REFERENCES

[1] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '06, (New York, NY, USA), pp. 93–102, Association for Computing Machinery, 2006.

[2] M. Bawa, T. Condie, and P. Ganesan, "Lsh forest: self-tuning indexes for similarity search," in *In WWW*, 2005.

[3] W. Jo, S. Kim, C. Lee, and T. Shon, "Packet preprocessing in cnn-based network intrusion detection system," *Electronics*, vol. 9, no. 7, p. 1151, 2020.

[4] Cisco, "Classifying network traffic," *QoS Classification Configuration Guide, Cisco IOS XE Release 3S*, vol. 2, p. 14, December 2018.

[5] K. L. Dias, M. A. Pongelupe, W. M. Caminhas, and L. de Errico, "An innovative approach for real-time network traffic classification," *Computer Networks*, vol. 158, pp. 143–157, 2019.

[6] M. Boger, T. Liu, J. Ratliff, W. Nick, X. Yuan, and A. Esterline, "Network traffic classification for security analysis," in *SoutheastCon 2016*, pp. 1–2, 2016.

[7] D.-Y. Kao, "Using the actionable intelligence approach for the dpi of cybercrime insider investigation," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pp. 1218–1224, 2020.

[8] A. Habibi Lashkari, G. Kaur, and A. Rahali, "Didarknet: A contemporary approach to detect and characterize the darknet traffic using deep image learning," in *2020 the 10th International Conference on Communication and Network Security*, ICCNS 2020, (New York, NY, USA), pp. 1–13, Association for Computing Machinery, 2021.

[9] P. Choorod and G. Weir, "Tor traffic classification based on encrypted payload characteristics," in *2021 National Computing Colleges Conference (NCCC)*, pp. 1–6, 2021.

[10] P. Perera, Y.-C. Tian, C. Fidge, and W. Kelly, "A comparison of supervised machine learning algorithms for classification of communications network traffic," in *International Conference on Neural Information Processing*, pp. 445–454, Springer, 2017.

[11] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *CoRR*, vol. abs/1709.02656, 2017.

[12] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pp. 103–114, 2011.

[13] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *2016 IEEE 24th International conference on network protocols (ICNP)*, pp. 1–5, IEEE, 2016.

[14] Z. Cao, G. Xiong, Y. Zhao, Z. Li, and L. Guo, "A survey on encrypted traffic classification," in *International Conference on Applications and Techniques in Information Security*, pp. 73–81, Springer, 2014.

[15] H. A. H. Ibrahim, O. R. A. Al Zuobi, M. A. Al-Namari, G. MohamedAli, and A. A. A. Abdalla, "Internet traffic classification using machine learning approach: Datasets validation issues," in *2016 Conference of Basic Sciences and Engineering Studies (SGCAC)*, pp. 158–166, IEEE, 2016.

[16] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in *2017 International Symposium on Wireless Communication Systems (ISWCS)*, pp. 1–6, IEEE, 2017.

[17] G. Sun, T. Chen, Y. Su, and C. Li, "Internet traffic classification based on incremental support vector machines," *Mobile Networks and Applications*, vol. 23, no. 4, pp. 789–796, 2018.

[18] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *ICISSP*, 2017.

[19] S. Cui, J. Liu, C. Dong, Z. Lu, and D. DU, "Only header: A reliable encrypted traffic classification framework without privacy risk," 2022.

[20] Z. Zou, J. Ge, H. Zheng, Y. Wu, C. Han, and Z. Yao, "Encrypted traffic classification with a convolutional long short-term memory neural network," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 329–334, 2018.

[21] IANA, "Assigned protocol numbers," February 2021.

[22] E. P. Freire, A. Ziviani, and R. M. Salles, "Detecting skype flows in web traffic," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, pp. 89–96, IEEE, 2008.

[23] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "Transport layer identification of p2p traffic," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, (New York, NY, USA), pp. 121–134, Association for Computing Machinery, 2004.

[24] M. Smith and R. Hunt, "Network security using nat and napt," in *Proceedings 10th IEEE International Conference on Networks (ICON 2002). Towards Network Superiority (Cat. No.02EX588)*, pp. 355–360, 2002.

[25] L. Costa, "Open systems interconnect (osi) model," *JALA: Journal of the Association for Laboratory Automation*, vol. 3, no. 1, pp. 28–35, 1998.

[26] J. Zhao, X. Jing, Z. Yan, and W. Pedrycz, "Network traffic classification for data fusion: A survey," *Information Fusion*, vol. 72, pp. 22–47, 2021.

[27] M. Hasanzadeh, H. Hamidi, and H. Asghari, "Plaintext transmission over session initiation protocol," in *7'th International Symposium on Telecommunications (IST'2014)*, pp. 629–634, 2014.

[28] K. A. Ogudo, "Analyzing generic routing encapsulation (gre) and ip security (ipsec) tunneling protocols for secured communication over public networks," in *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pp. 1–9, 2019.

[29] S. Zhang, A. Li, H. Zhu, Q. Sun, M. Wang, and Y. Zhang, "Research on the protocols of vpn," in *Advances in Intelligent Systems and Interactive Applications* (F. Xhafa, S. Patnaik, and A. Y. Zomaya, eds.), (Cham), pp. 554–559, Springer International Publishing, 2018.

[30] D. Maimon, M. Becker, S. Patil, and J. Katz, "Self-protective behaviors over public wifi networks," in *The LASER Workshop: Learning from Authoritative Security Experiment Results (LASER 2017)*, pp. 69–76, USENIX Association, October 2017.

[31] M. Capellupo, J. Liranzo, M. Z. A. Bhuiyan, T. Hayajneh, and G. Wang, "Security and attack vector analysis of iot devices," in *Security, Privacy, and Anonymity in Computation, Communication, and Storage* (G. Wang, M. Atiquzzaman, Z. Yan, and K.-K. R. Choo, eds.), (Cham), pp. 593–606, Springer International Publishing, 2017.

[32] Y. Wang, E. Kjerstad, and B. Belisario, "A dynamic analysis security testing infrastructure for internet of things," in *2020 Sixth International Conference on Mobile And Secure Services (MobiSecServ)*, pp. 1–6, 2020.

[33] M. El, E. McMahon, S. Samtani, M. Patton, and H. Chen, "Benchmarking vulnerability scanners: An experiment on scada devices and scientific instruments," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 83–88, 2017.

[34] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Passive and Active Network Measurement* (S. Uhlig, K. Papagiannaki, and O. Bonaventure, eds.), (Berlin, Heidelberg), pp. 165–175, Springer Berlin Heidelberg, 2007.

[35] E. Papadogiannaki and S. Ioannidis, "Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware," *Sensors*, vol. 21, no. 4, 2021.

[36] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," 2013.

[37] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for qos: A statistical signature-based approach to ip traffic classification," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, (New York, NY, USA), pp. 135–148, Association for Computing Machinery, 2004.

[38] V. Paxson, "Empirically derived analytic models of wide-area tcp connections," *IEEE/ACM Transactions on Networking*, vol. 2, no. 4, pp. 316–336, 1994.

[39] O. Salman, I. H. Eljhajj, A. Kayssi, and A. Chehab, "A review on machine learning-based approaches for internet traffic classification," *Ann. Telecommun.*, vol. 75, pp. 673–710, December 2020.

[40] J. Cao, Z. Fang, G. Qu, H. Sun, and D. Zhang, "An accurate traffic classification model based on support vector machines," *Netw.*, vol. 27, p. n/a, January 2017.

[41] H.-K. Lim, J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong, and Y.-H. Han, "Packet-based network traffic classification using deep learning," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 046–051, 2019.

[42] Z. Li, R. Yuan, and X. Guan, "Accurate classification of the internet traffic based on the svm method," in *2007 IEEE International Conference on Communications*, pp. 1373–1378, 2007.

[43] M. Song, J. Ran, and S. Li, "Encrypted traffic classification based on text convolution neural networks," in *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 432–436, 2019.

[44] Z. Yuan and C. Wang, "An improved network traffic classification algorithm based on hadoop decision tree," in *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, pp. 53–56, 2016.

[45] E. L. Goodman, C. Zimmerman, and C. Hudson, "Packet2vec: Utilizing word2vec for feature extraction in packet data," *CoRR*, vol. abs/2004.14477, 2020.

[46] Y. Wang, Y. Xiang, and S.-Z. Yu, "Automatic application signature construction from unknown traffic," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 1115–1120, 2010.

[47] R. Fielding, "Hypertext transfer protocol – http/1.1," rfc, IETF, June 1999.

[48] J. R. J. Postel, "File transfer protocol (ftp)," rfc, IETF, October 1985.

[49] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core," rfc, IETF, March 2011.

[50] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (USA), p. 19, USENIX Association, 2004.

[51] Y. Tang, B. Xiao, and X. Lu, "Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms," *Computers & Security*, vol. 28, no. 8, pp. 827–842, 2009.

[52] B. D. Sija, K.-S. Shim, and M.-S. Kim, "Automatic payload signature generation for accurate identification of internet applications and application services," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 4, pp. 1572–1593, 2018.

[53] Y. Wang, Y. Xiang, W. Zhou, and S. Yu, "Generating regular expression signatures for network traffic classification in trusted network management," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 992–1000, 2012. Special Issue on Trusted Computing and Communications.

[54] G. Szabô, Z. Turânyi, L. Toka, S. Molnâr, and A. Santos, "Automatic protocol signature generation framework for deep packet inspection," ICST, 6 2012.

[55] B.-C. Park, Y. J. Won, M.-S. Kim, and J. W. Hong, "Towards automated application signature generation for traffic identification," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, pp. 160–167, 2008.

[56] M. Ye, K. Xu, J. Wu, and H. Po, "Autosig-automatically generating signatures for applications," in *2009 Ninth IEEE International Conference on Computer and Information Technology*, vol. 2, pp. 104–109, 2009.

[57] C. VinothGeorge and V. Ewards, "Efficient regular expression signature generation for network traffic classification," 2013.

[58] H. Toivonen, *Apriori Algorithm*, pp. 60–60. Boston, MA: Springer US, 2017.

[59] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, (San Francisco, CA, USA), pp. 487–499, Morgan Kaufmann Publishers Inc., 1994.

[60] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[61] S. Altschul, *basic local alignment search tool (BLAST)*. online, 2003.

[62] Intel Software, “Hyperscan.”

[63] X. Wang, Y. Hong, H. Chang, K. Park, G. Langdale, J. Hu, and H. Zhu, “Hyperscan: A fast multi-pattern regex matcher for modern cpus,” in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, (Boston, MA), pp. 631–648, USENIX Association, Feb. 2019.

[64] tommyod, “efficient-apriori.”

[65] KimiNewt, “Pyshark.”

[66] NumPy, “Numpy.”

[67] Rene Rivera, Bernan Dawes, David Abrahams, “Boost.”

[68] M. Kapoor, G. Fuchs, and J. Quance, “Rexactor: Automatic regular expression signature generation for stateless packet inspection,” in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, pp. 1–9, 2021.

[69] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications.” RFC 3550, july 2003.

[70] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, and J. M. Smith, *DeepMatch: Practical Deep Packet Inspection in the Data Plane Using Network Processors*, pp. 336–350. New York, NY, USA: Association for Computing Machinery, 2020.

[71] Google, “Google transparency report,” 2022.

[72] Z. Fu, Z. Liu, and J. Li, “Efficient parallelization of regular expression matching for deep inspection,” in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, 2017.

[73] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, “A survey on locality sensitive hashing algorithms and their applications,” 2021.

[74] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC ’98, (New York, NY, USA), pp. 604–613, Association for Computing Machinery, 1998.

[75] A. Z. Broder, “Identifying and filtering near-duplicate documents,” in *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, COM ’00, (Berlin, Heidelberg), pp. 1–10, Springer-Verlag, 2000.

[76] M. Gabryel, K. Grzanek, and Y. Hayashi, “Browser fingerprint coding methods increasing the effectiveness of user identification in the web traffic,” *Journal of Artificial Intelligence and Soft Computing Research*, vol. 10, no. 4, pp. 243–253, 2020.

[77] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering.," in *NDSS*, vol. 9, pp. 8–11, Citeseer, 2009.

[78] P. Lee, L. V. Lakshmanan, and J. X. Yu, "On top-k structural similarity search," in *2012 IEEE 28th International Conference on Data Engineering*, pp. 774–785, 2012.

[79] H. Yang, Q. He, Z. Liu, and Q. Zhang, "On top-k structural similarity search," *Security and Communication Networks*, vol. 2021, 2021.

[80] Z. Tang, Q. Wang, W. Li, H. Bao, F. Liu, and W. Wang, ""hslf: Http header sequence based lsh fingerprints for application traffic classification"," in *Paszynski M., KranzlmÃŒller D., Krzhizhanovskaya V.V., Dongarra J.J., Sloot P.M.A. (eds) Computational Science - ICCS 2021*, 2021.

[81] W. Jiang and M. Gokhale, "Real-time classification of multimedia traffic using fpga," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 56–63, 2010.

[82] Y. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets.* Standford University, 2022.

[83] B. Sangster, T. J. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, and G. Conti, "Toward instrumenting network warfare competitions to generate labeled datasets," in *Proceedings of the 2nd Conference on Cyber Security Experimentation and Test*, CSET'09, (USA), p. 9, USENIX Association, 2009.

[84] C. Guarnieri, "Skynet: a tor-powered botnet straight from reddit," 2012.

[85] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani, "VPN/non-VPN Dataset (ISCXVPN2016); Tor/non-Tor Dataset (ISCXTor2016)," 2016.

[86] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani, "Tor/non-Tor Dataset (ISCXTor2016)," 2016.

[87] J. V. V. Silva, N. R. Oliveira, D. S. V. Medeiros, M. A. Lopez, and D. M. F. Mattos, "A statistical analysis of instrinsic bias of network security datasets for training machine learning mechanisms," February 2022.

[88] K. Tumer and J. Ghosh, "Error correlation and error reduction in ensemble classifiers," *Connection science*, vol. 8, no. 3-4, pp. 385–404, 1996.

[89] TeamStage, "Zoom statistics 2022: How video conferencing changed our lives," 2022.

[90] D. Curry, "Microsoft teams revenue and usage statistics (2022)," June 2022.

[91] G. W. Edwards, M. J. Gonzales, and M. A. Sullivan, "Robocalling: Stirred and shaken! - an investigation of calling displays on trust and answer rates," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, (New York, NY, USA), pp. 1–12, Association for Computing Machinery, 2020.

[92] Z. Z. Wu, J. Guo, C. Zhang, and C. Li, "Steganography and steganalysis in voice over ip: A review," *Sensors (Basel, Switzerland)*, vol. 21, 2021.

[93] S. Nagaraja and R. Shah, "Voiploc: Voip call provenance using acoustic side-channels," *IEEE Security and Privacy 2020*, 2019.

[94] C. Choti, N. Hnoohom, S. Tritilanunt, and S. Yuenyong, "Prediction of intrusion detection in voice over internet protocol system using machine learning," in *The 2021 9th International Conference on Computer and Communications Management*, pp. 149–155, 2021.

[95] M. Kmet, P. Matousek, and O. R. Martin, "Fast rtp detection and codecs classification in internet traffic," 2014.

[96] M. Sha, T. Manesh, and S. M. A. El-atty, "Voip forensic analyzer," *International Journal of Advanced Computer Science and Applications*, vol. 7, 2016.

[97] D.-Y. Kao, T.-C. Wang, and F.-C. Tsai, "Forensic artifacts of network traffic on wechat calls," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pp. 262–267, IEEE, 2020.

[98] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[99] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv*, vol. abs/1511.08458, 2015.

[100] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[101] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[102] H.-K. Lim, J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong, and Y.-H. Han, "Packet-based network traffic classification using deep learning," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 046–051, 2019.

[103] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3274–3280, 2019.

[104] J.-L. Costeux, F. Guyard, and A.-M. Bustos, “Qrp08-5: Detection and comparison of rtp and skype traffic and performance,” in *IEEE Globecom 2006*, pp. 1–5, 2006.

[105] N. Patwari, A. O. Hero III, and A. Pacholski, “Manifold learning visualization of network traffic data,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pp. 191–196, 2005.

[106] Y. Wang and J. M. Solomon, “Deep closest point: Learning representations for point cloud registration,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[107] M. Quach, G. Valenzise, and F. Dufaux, “Folding-based compression of point cloud attributes,” in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 3309–3313, 2020.

[108] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “Dbscan revisited, revisited: why and how you should (still) use dbscan,” *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[109] N. Rahmah and I. S. Sitanggang, “Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra,” in *IOP conference series: earth and environmental science*, vol. 31, p. 012012, IOP Publishing, 2016.

[110] T. Mullin, “Dbscan parameter estimation using python,” july 2020.

[111] H. Song and J. W. Lockwood, “Efficient packet classification for network intrusion detection using fpga,” in *FPGA ’05*, 2005.

[112] C. Wang, M. Ji, J. Wang, W. Wen, T. Li, and Y. Sun, “An improved dbscan method for lidar data segmentation with automatic eps estimation,” *Sensors*, vol. 19, no. 1, 2019.

[113] T. G. Dietterich, “Ensemble methods in machine learning,” in *International workshop on multiple classifier systems*, pp. 1–15, Springer, 2000.

[114] L. A. Iliadis and T. Kaifas, “Darknet traffic classification using machine learning techniques,” in *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pp. 1–4, 2021.

[115] H. Ma, J. Cao, B. Mi, D. Huang, Y. Liu, and Z. Zhang, “Dark web traffic detection method based on deep learning,” in *2021 IEEE 10th Data Driven Control and Learning Systems Conference (DDCLS)*, pp. 842–847, 2021.

[116] D. Sarkar, P. Vinod, and S. Y. Yerima, “Detection of tor traffic using deep learning,” in *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*, (Los Alamitos, CA, USA), pp. 1–8, IEEE Computer Society, nov 2020.

[117] G. R. S. Weir, "The posit text profiling toolset," 2007.

[118] D. Wood, N. Apthorpe, and N. Feamster, "Cleartext data transmissions in consumer iot medical devices," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pp. 7–12, 2017.

[119] W. M. Shbair, T. Cholez, J. FranÃ§ois, and I. Chrisment, "A survey of https traffic and services identification approaches," *ArXiv*, vol. abs/2008.08339, 2020.

[120] A. P. Singh and M. Singh, "A comparative review of malware analysis and detection in https traffic," *International Journal of Computing and Digital Systems*, vol. 10, no. 1, pp. 111–123, 2021.

[121] F. Casino, K.-K. R. Choo, and C. Patsakis, "Hedge: Efficient traffic classification of encrypted and compressed packets," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 11, pp. 2916–2926, 2019.

[122] D. Hahn, N. J. Apthorpe, and N. Feamster, "Detecting compressed cleartext traffic from consumer internet of things devices," *ArXiv*, vol. abs/1805.02722, 2018.

[123] M. Zain ul Abideen, S. Saleem, M. Ejaz, and R. Soundrapandiyan, "Vpn traffic detection in ssl-protected channel," *Sec. and Commun. Netw.*, vol. 2019, jan 2019.

[124] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "Sshcure: A flow-based ssh intrusion detection system," in *Autonomous Infrastructure, Management and Security*, 2012.

[125] V. Ghiëtte, H. J. Griffioen, and C. Doerr, "Fingerprinting tooling used for ssh compromisation attempts," in *International Symposium on Recent Advances in Intrusion Detection*, 2019.

[126] S. Kerr, L. Song, and R. Wan, "A review of DNS over port 80/443," Internet-Draft draft-shane-review-dns-over-http-04, Internet Engineering Task Force, November 2016. Work in Progress.

[127] M. Simioni, P. Gladyshev, B. Habibnia, and P. R. Nunes de Souza, "Monitoring an anonymity network: Toward the deanonymization of hidden services," *Forensic Science International: Digital Investigation*, vol. 38, p. 301135, 2021.

[128] M. MontazeriShatoori, L. Davidson, G. Kaur, and A. Habibi Lashkari, "Detection of doh tunnels using time-series classification of encrypted traffic," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pp. 63–70, 2020.

[129] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)." RFC 8484, Oct. 2018.

[130] QuoIntelligence, "How dns-over-https (doh) has changed the threat landscape for companies," February 2021.

[131] TrendMicro, "New godlua backdoor found abusing dns over https (doh) protocol," July 2019.

[132] P. T. I. Team, September 2019.

[133] W. Wong, "Stunnel: Ssling internet services easily," *SANS Institute, November*, 2001.

[134] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer, "Hello from the other side: Ssh over robust cache covert channels in the cloud," in *Network and Distributed System Security Symposium*, 2017.

[135] M. Kapoor, S. Krishnan, and T. Moyer, "Deep packet inspection at scale: Search optimization through locality-sensitive hashing," in *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, vol. 21, pp. 97–105, 2022.

[136] K. Zhou, W. Wang, C. Wu, and T. Hu, "Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks," *Etri Journal*, vol. 42, pp. 311–323, 2020.

[137] B. Ricks, P. Tague, and B. Thuraisingham, "Ddos-as-a-smokescreen: Leveraging netflow concurrency and segmentation for faster detection," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 217–224, 2021.

[138] F. D. Gaspari, D. Hitaj, G. Pagnotta, L. D. Carli, and L. V. Mancini, "Encod: Distinguishing compressed and encrypted file fragments," in *International Conference on Network and System Security*, pp. 42–62, Springer, 2020.

APPENDIX A: QUADRATIC FIT COMPARISON GRAPHS

The appendices should be used for whatever material you or your advisory committee believes should be included, but would not be appropriate in the text of the thesis or dissertation. Such materials can include:

1. the original data obtained in the thesis or dissertation research, including computer programs and printouts, surveys, or correspondence;
2. detailed descriptions of procedures, which go beyond the general outline of methods and approaches presented in the text;
3. a particularly extensive review of the literature and other information that may be useful to future scholars who may wish to delve more deeply into the research topic.

A.1 Section in appendix

This is a section in the appendix.