A Framework for Classifying Applications from Raw Network Traffic Traces

Alireza Marefat, Abbaas Alif Mohamed Nishar, Ashwin Ashok

[amarefatvayghani1, amohamednishar1]@student.gsu.edu,
aashok@gsu.edu

Georgia State University, Atlanta, USA

***Abstract*—In this paper, we study and present the design of a framework to identify applications from raw network traces. Framing the problem as an application classification problem, we set up the framework to extract key features from packet data and their temporal behavior. The feature generation, their training using traditional machine learning models, and the decision making are executed over a four- stage pipeline, to yield the name of the application. Through an in-lab environment experimentation using OpenWrt toolkit, RaspberryPi, and a set of physical devices (generating network traffic), we evaluated on average about 204K data points from the captured network packet traces for six applications. Our results show that our method is able to classify the applications with at least 90% accuracy. Through micro-benchmarking, we also show the feasibility of scaling the number of applications and running the tool in real-time.**

***Index Terms*—Network Traffic Classification, Application Classification, Feature Extraction, Software-Defined Network, Wireless Network, Machine Learning**

I. Introduction

Extensive research has been conducted on Network Traffic Classification (NTC) since the late 2000s. The initial focus was on developing innovative statistical tools to characterize broad traffic classes and specific applications within each class. This aimed to replace traditional methods such as light and Deep Packet Inspection (DPI), including port-based and payload-based classification. Seminal works like [1], [2] sparked the first wave of approaches, employing classic Machine Learning (ML) to classify services through basic feature engineering. This wave culminated in straightforward yet effective "early traffic classification" techniques, which used time series information, such as the size, ports, and protocols, for decision-making. Subsequently, the success of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) in image recognition [3] led to a second wave of traffic classification approaches utilizing Deep Learning (DL) techniques which is not the emphasis on this paper.

While DL techniques achieve the best accuracy in traffic classification, conventional ML algorithms still exhibit some advantages over DL. First, conventional ML algorithms are relatively lightweight and thus are able to achieve real-time traffic classification. However, the computational complexity of DL algorithms is high and this can pose difficulties in achieving real-time performance in high-speed networks. Second, DL techniques often require a large amount of labeled data to achieve satisfactory accuracy. However, obtaining a large labeled dataset is expensive and time-consuming [4]. Hence, our focus will be more on the first wave, as we believe that a comprehensive set of features has not been thoroughly considered and accurately analyzed thus far. Despite numerous NTC works using ML, there remains uncertainty about which features are more important than others.

Limitations of traditional NTC. Traditional Network Traffic Classification (NTC) methods face limitations, primarily relying on rule-based approaches to classify network ports, packet payload, traffic statistics, and application behavior. Port-based classification, associating specific port numbers with applications, is increasingly unreliable due to the use of non-standard or dynamically allocated ports [5]. Payload-based classification, effective for unique characteristics but ineffective for encrypted traffic [6]. Statistics-based classification, while flexible, demands extensive data and computational resources [7]. Behavior-based classification, focusing on unique application behaviors, requires a deep understanding, resulting in high implementation complexity [8].

Proposed approach. In this paper, we design, implement, and evaluate an application classification system that leverages machine learning. In addition, our methodology combines deep packet inspection, recognizing the challenges in traditional NTC, we advocate for a hybrid approach, particularly packet-based, payload-based, statistical-based, and behavioral-based methods, in a multi-level machine learning architecture. The objective is to demonstrate the superiority of multidimensional features over rule-based traditional NTC, emphasizing feature importance. Our design targets to achieve high classification accuracy with high computation efficiency, potentially scalable and function real-time. using a hybrid approach for NTC.

In summary, the contributions of this paper are as follows:

(1) A unique four-phased architecture for application-centric NTC, leveraging machine learning. The model includes feature extraction, feature vector generation, and multi-level machine learning, utilizing Naive Bayes Multinomial for coarse classification and Random Forest and Decision Tree for fine classification.

(2) We design and implement a software-defined wireless

SoutheastCon 2024 | 979-8-3503-1710-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/SOUTHEASTCON52093.2024.10500148

network (SDWN) using Open-Wrt [9], simulating a realistic home/office programmable wireless network. Our SDWN serves as a tool for generating, capturing, and managing traffic for network traffic classification research, enhancing sandboxing in network traffic studies.

(3) Our research introduces a hybrid NTC method combining packet-based, payload-based, statistical-based, and behavioral-based network features. This includes a blend of packet and payload lengths, flow-level features, nominal parameters, and statistical distributions. In addition, a comprehensive analysis using diagrams (circle packing, and word cloud) to illustrate application distinctions based on network flow, types, destination ports, protocols, DNS, TLS/SSL requests, and feature importance.

(4) We contribute a rich dataset with over 204,000 data points from experiments involving six applications. The dataset includes popular services like YouTube, Spotify, WhatsApp, Wyze IPcamera, Google Home, email, and web browsing. Three additional applications (Amazon Prime, Castbox, and Discord) are added for scalability evaluation.

(5) Our extensive experiment-based evaluation reports 90% to 100% accuracy on application-level classification across different machine learning training data volumes and capture time windows. The system exhibits consistent results for scalability and real-time deployment settings.

II. Related Works

In this section, we discuss some works that focus on NTC in wireless networks. Most of the related works in NTC are about device identification and classification however some works are somehow related to application classification. In the following, we will discuss the most related works in which they implemented the NTC using ML and DL methods using different features.

In [10], Bezawada et al. introduces a technique for IoT device behavioral fingerprinting, leveraging network traffic features for device identification. An ML model, trained using these features (i.e. TCP window size, entropy, and payload lengths), can distinguish similar device types. The study [11] proposed a context-aware traffic classification approach employing ML classifiers to enhance WLAN power efficiency. The authors captured real-time instances of some applications including Skype, Google Hangouts, Facebook, Gmail, New Star Soccer, and XiiaLive internet radio. They employed 6 features per instance like receiving data rate, transmitting data rate, total received kbytes, total transmitted Kbytes, total number of received packets, and total number of transmitted packets. They used different ML techniques to evaluate their work. Sivanathan et al. in [12] proposed a 2-stage ML framework that utilizes various network traffic characteristics to identify and classify baseline behavior of IoT devices in their instrumented smart environment. The broad range of traffic characteristics studied includes activity patterns (e.g., distribution of volume/times during active/sleep periods), and signaling (e.g., domain names requested, server-side port numbers used and TLS handshake exchanges). The statistical attributes used in this work consist of activity cycles, port numbers, signaling patterns, and cipher suites. In the paper [13], the authors leverage supervised ML models for network traffic classification in a Software Defined Networking-enabled Fiber-Wireless Internet of Things smart environment. The approach improves network interoperability, reliability, scalability, and facilitates enhanced resource allocation and network security. The paper [14] formulated a method that transforms packet headers into time-series feature vectors, and subsequently into pseudo-images, enabling the application of CNN for traffic classification. Utilized features are source port, destination port, the number of bytes in packet payload, TCP window size, inter-arrival time and direction of the packet. The paper [15] introduces a novel approach to classify encrypted network traffic and identify applications, without relying on deep packet inspection. The authors transform basic flow telemetry data into intuitive FlowPic representations that encapsulate timing and size attributes. These image-like fingerprints of traffic patterns are then leveraged to train CNN models for classification tasks. As for the features, they only considered packet size and arrival time of the packets. Evaluations demonstrate high accuracy in categorizing encrypted VPN and Tor traffic into browsing, chat, video and other flow types. In [16], Camelo et al. introduce a spectrum-based approach for traffic classification across the radio network stack, realized via a Deep Learning-based classifier. About 140K samples are gathered from six applications including Spotify, Tunein, Gpodcast, Youtube, Netflix, and Twitch. The paper compares the performance of a novel CNN and a RNN architecture for traffic classification tasks.

Although the discussed related works utilized various approaches for NTC, none of them considered as diverse set of features as we do. One of the boldest novelties of this work is employing a hybrid NTC methods.

III. System Architecture Design Overview

As illustrated in Fig. 1, we propose a four-phase system design featuring a 2-level machine learning hierarchical architecture. The notion of such a design is to keep the architecture modular while ensuring each stage acts as a *filter* to simplify the problem or derive key parameters or *features* deliverable to the next phase. At a high level, Phase 1 qualifies the *Traffic Analysis* (section V) aspect followed by the *Multi-level Machine Learning* (section VI) that compiles Phases 2, 3 and 4.

A. Phase-1 (Feature Extraction)

Packet capture (PCAP), also referred to as libpcap, serves as an application programming interface (API) capturing live network packet data spanning OSI model Layers 2-7. Fig. 2 illustrates a segment of the raw PCAP file captured in our setup ((section IV) provides a detailed discussion). The raw PCAP data lacks sufficient information for traffic classification, as it only reveals headers and a limited amount of encrypted/unencrypted payloads. Consequently, feature extraction is necessary to isolate the specific features we require from the raw PCAPs.

Hence, we leverage the benefits of our hybrid NTC approachas we considered packet-based, payload-based, statistical-based, and behavioral-based methods to extract the relevant key features for application classification. Defining

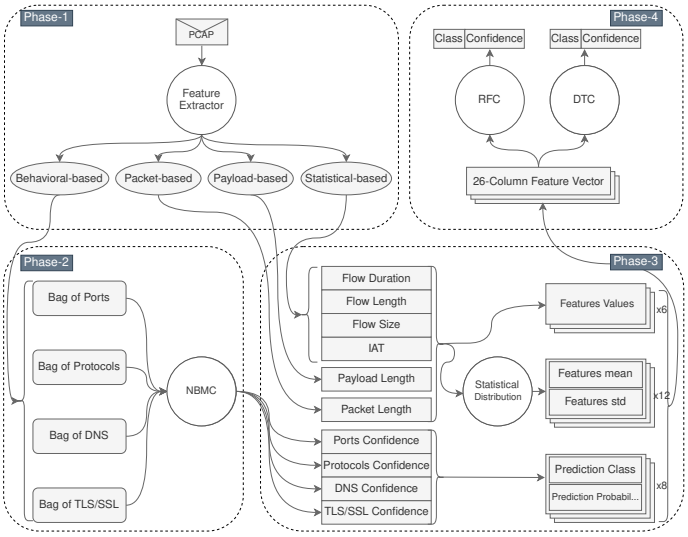


Fig. 1. Proposed system architecture for application classification using network traces.

network flow as a "5-tuple", encompassing the {IP source, IP destination, source port, destination port, and protocol}, we check on each row in the `DataFrame ('df')`, each representing a network packet, and aggregate these packets into flows. We extract four flow-based features: "*Flow Duration*" (time difference between the first and last packet in a flow), "*Flow Length*" (count of packets in a flow), "*Flow Size*" (sum of lengths of all packets in a flow), and "*Inter-Arrival Time*" (time difference between last packet of previous flow and first packet of current flow). Next, we extract packet attributes, such as "*packet Length*" and "*Payload Length*". As a behavioral-based approach, we consider four attributes including "*ports*", "*protocols*", "*DNS*" and "*TLS/SSL*" queries. These are non-numerical and are treated as discrete entities. These nominal attributes are handled using a Bag of Words method, which we refer to as a 'Bag of Features'. The output of nominal parameters will be used in Phase-2 (as bags of features model) for coarse classification, while the output of the continuous parameters will be utilized in Phase-3 for statistical analysis.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.923	Raspberr_30:61:fe	IntelCor_da:64:75	ARP	71	Who has 192.168.2.192? Tell 192.168.2.1
2	1.219	54.235.180.207	192.168.2.192	TLSv1.2	176	Application Data
3	1.279	192.168.2.192	54.235.180.207	TCP	54	54625 > 443 [ACK] Seq=55 Ack=57 Win=257 Len=0
4	1.386	20.62.59.38	192.168.2.192	TLSv1.2	81	Application Data
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 2. Raw PCAP file (Indicating the first four captured packets in a raw format). Each row is a network traffic analysis entry.

B. Phase-2 (ML Level-1 [Coarse Classification])

The feature vector for ports, protocols, DNS queries, and TLS/SSL queries are considered *nominal*, which means they are not considered numeric values and are multi-valued. To elaborate, 'ports' represents a collection of destination ports used by a single application throughout the duration of packet capture. Our proposed system model operates on a two-tier ML architecture. Initially, each nominal attribute is presented to its corresponding level-1 classifier (Naive Bayes Multinomial) in a bag-of-features format. This approach generates a matrix wherein rows denote labeled instances and columns correspond to unique features. This 'M x N' matrix (where 'M' is the total number of instances and 'N' is the number of unique features) underpins the second phase of our model. The output of this classifier comprises a class of confidence for each attribute, yielding both the predicted class and the prediction probability. These four outputs are subsequently amalgamated with other features and incorporated into the fourth phase of our system model as an additional input for the level-2 classifiers.

C. Phase-3 (Statistical Distribution + Final Feature Vector)

During this phase, we employ statistical distribution metrics, specifically the mean and standard deviation (std), for the six features derived from Phase-1. Consequently, each feature will now be represented by three distinct values: the raw value, the mean, and the standard deviation. For generating the final feature vector, we use the raw values for "Flow Duration", "Flow Length", "Flow Size", and "Inter Arrival Time (IAT)" in addition to "packet length" and "payload length". Following this, statistical distribution parameters such as mean and standard deviation for each of these six features are incorporated into the `'df'`. Subsequently, bags of feature confidences, including "predicted class" and "Prediction probability" for DNS, Port, Protocol, and TLS/SSL, are merged into the `'df'`. Consequently, the consolidated feature vector used in this phase comprises 26 columns.

D. Phase-4 (ML level-2 [Fine Classification])

This constitutes the final phase, where application classification is accomplished. The consolidated feature vector serves as the input for two ML classifier algorithms. Initially, the feature vector is supplied to the Random Forest Classifier (RFC) [17], generating an output consisting of both the confidence score and the predicted class. As an alternative for comparison, we also repeat the same methodology with the Decision Tree Classifier (DTC) [18] and obtain the corresponding classification output.

IV. Experimentation Setup and Data Capture

Before delving into traffic analysis and ML classification details, we outline our empirical experimentation and data collection process. Our methodical experiments aimed to generate a valuable dataset for system development and performance evaluation. For design purposes, we employed a small subset of the dataset, reserving the majority for a fair evaluation. **Our design assumes that during each data capture window, only one application from each device is considered, reflecting the naturally sequential flow of network traffic into a router.**

A. Experimental Setup

SDWN is a wireless network architecture enabling intelligent and centralized control through software applications for monitoring and management. It adopts a Software-Defined Networking (SDN) paradigm, where the control plane is separated from the data plane, allowing centralized control and programmability [19]. Our research applies this SDWN approach to a practical home/office wireless network topology, integrating both IoT and non-IoT devices. This facilitates a programmable control plane for dynamic network management,

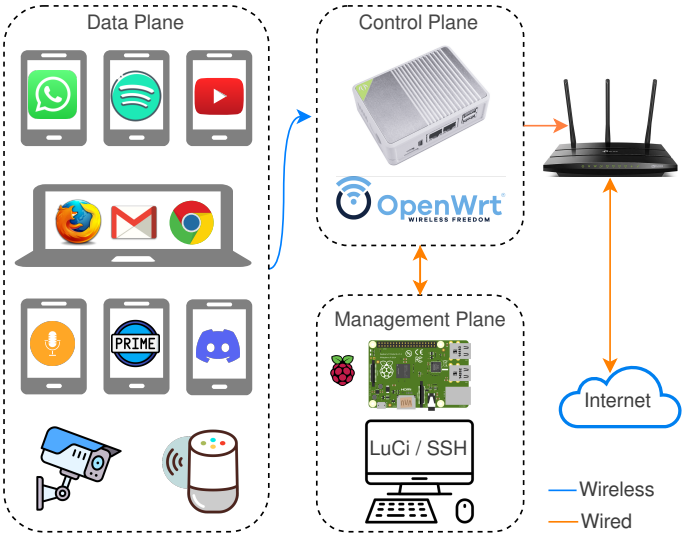


Fig. 3. Our software-defined wireless networking experimentation setup

laying the groundwork for exploring the convergence of SDWN and network traffic monitoring. This integration provides the context for our investigation into application identification.

As depicted in Fig. 3, we used Seeed reRouter CM4 1432 [20], a Raspberry Pi Based Mini Router with OpenWrt operating system as a programmable control plane. The base hardware is a Raspberry pi 4. We transformed the TP-Link router into a dummy Wide Area Network (WAN) switch responsible solely for handling the data plane, while the control plane of our system was established on the OpenWrt device. The WAN interface of the TP-Link wireless router is linked to the public Internet through our organization's preconfigured network. OpenWrt is connected to the Local Area Network (LAN) ports of the router. The LAN subnet connecting the TP-Link router and OpenWrt is configured as 192.168.1.0/30. In this subnet, which uses a subnet mask of 30 (255.255.255.252), the maximum number of hosts is two, which suits our scenario. For the management plane to provide us with a console, we connected a Raspberry Pi 4.0 to the LAN port of the OpenWrt device. The internal network subnet is set as 192.168.2.0/24. Using a subnet mask of 29 (255.255.255.240), which provides us with enough hosts per subnet to accommodate the number of end nodes on the LAN, connected to this network. The Raspberry Pi provides us with both the graphical user interface (GUI) of OpenWrt, known as LuCi [21], and SSH access to the OpenWrt device with root privileges. Through the LuCi web interface, we configured the initial IEEE 802.11 WiFi network settings on OpenWrt. Additionally, on the LAN side, we connected six wireless devices to the OpenWrt access point: a Windows laptop, two Android phones, one Android tablet, one IP camera, and one Google Home device acting as our data plane.

Applications were chosen based on popularity and relevance to modern internet services to mimic a mixed IoT/non-IoT environment. The laptop runs browsing and email traffic, Android phones are used for WhatsApp calls, a tablet runs Spotify for audio streaming, a Wyze IP camera streams video, and a Google Home device focuses on audio streaming. The IP camera and Google Home are categorized as IoT devices. Discord, Castbox, and Amazon Prime are also considered, for evaluating the scalability of our solution.

B. Network Flow and Data Capture

A *network flow* starts from any end node that generates the traffic and sends it to the OpenWrt (acts as an Access Point (AP)). OpenWrt forwards it to the router and thus to the internet. For example, an Android tablet with the IP address 192.168.2.200 sends packets to OpenWrt (192.168.2.1), which are then forwarded to 192.168.1.1, and subsequently to Spotify server's public IP on the internet.

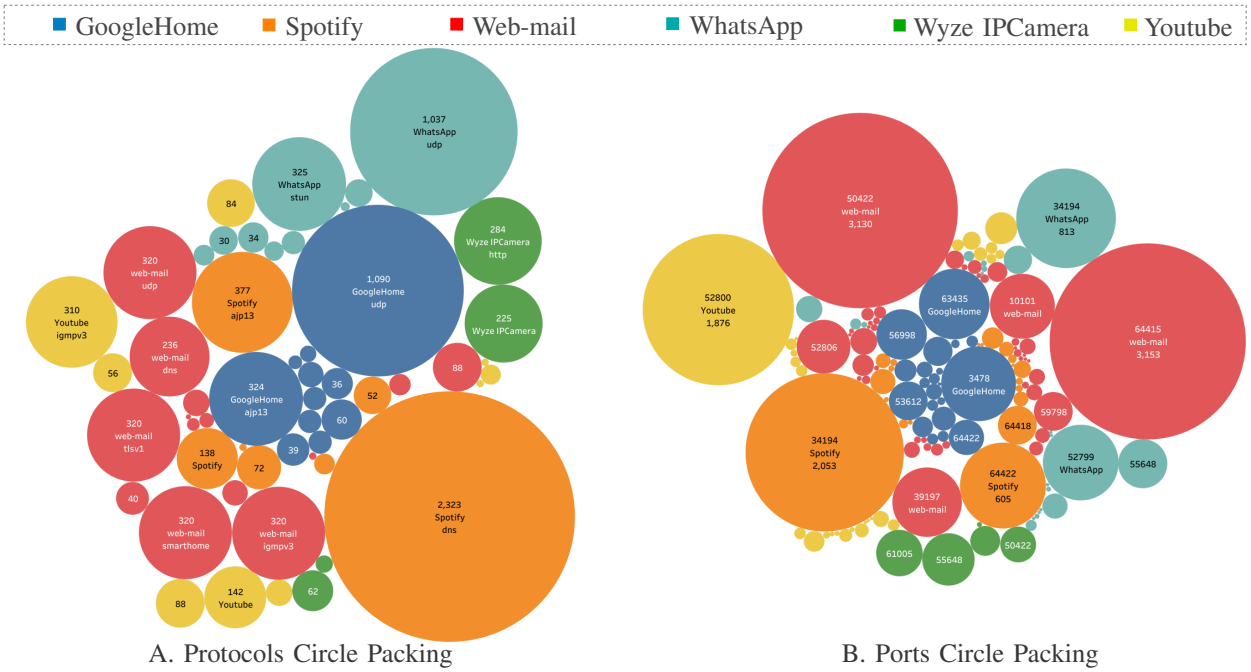
For network traffic capture, we utilized the `tcpdump` tool [22] on our controller, which saves the data in the PCAP format

The sample command executed on OpenWrt is `"tcpdump -vvv -n -i wlan0 -G 600 -U -s 0"`. Here, the '-vvv' verbose command enhances packet information retrieval. Each 'v' progressively magnifies detail. For example, one 'v' provides the TTL, ID, total length, and options in an IP packet, and three 'v's offer even more detailed data. The '-n' option avoids converting host addresses to names, mitigating potential delays and packet loss during captures. The '-i wlan0' specifies the listening interface, the wireless LAN interface of OpenWrt (192.168.2.1), and '-G 600' constrains the capture duration to 600 seconds. The '-s 0' option sets the packet capture size in bytes, with a size of 0 signifying full packet capture. After the capture, we use the WireShark Packet Analyzer [23] tool to transform the packet capture data from PCAP to easier-to-handle CSV format. The dataset is an aggregation of packet captures from several rounds of data collection.

V. Traffic Analysis

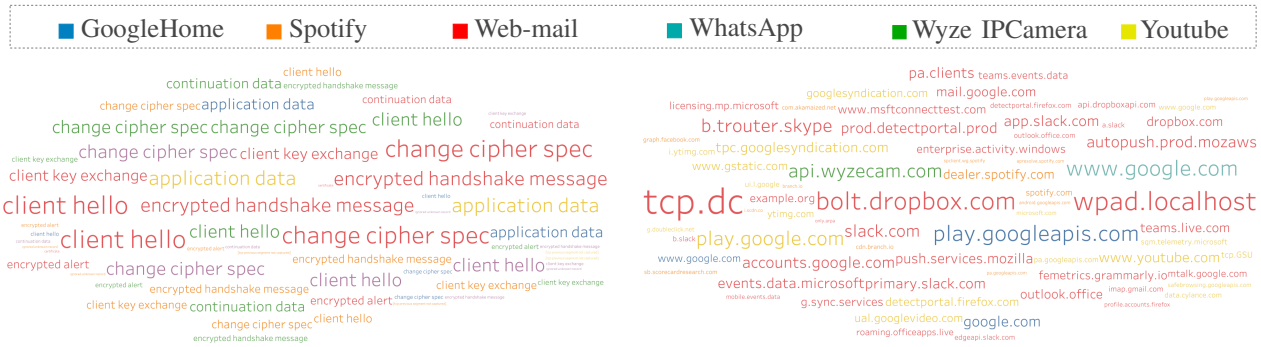
The objective of this stage is *feature extraction*, that is, to discern the varying behaviors of individual applications over time, unravel any underlying patterns, behaviors, or signatures that could be identified as unique characteristics inherent to specific applications. Applications typically make distinctive DNS requests, and identifying this patterns can help track the application. Destination ports, although not entirely reliable as they are variable, still are valuable, as different applications often use specific ports associated with their communication protocols. Protocols themselves are highly indicative of application types, given their close alignment with application functionalities. Lastly, the TLS/SSL handshake (authentication/security) components offer insight into the applications initiating secure connections, such as the Client Key Exchange, Change Cipher Spec, and Encrypted Handshake Message. Despite potential obfuscation and encryption challenges, combining these key features enhances our classification capabilities.

Protocols. Protocols can be key application defining factors [24]. TCP and UDP, the main transport layer protocols, are the most common and recurring in our dataset. TCP ensures reliable data delivery, typically for web browsing and email services, while UDP supports real-time communication [25], used by applications like WhatsApp or IP Cameras. Application



A. Protocols Circle Packing B. Ports Circle Packing

Fig. 4. Most used/cited Ports and Protocols per application represented in circle pack (better read in color - digital or print)



A. TLS/SSL Queries Word Cloud B. DNS Queries Word Cloud

Fig. 5. Most used/cited DNS and TLS/SSL queries per application represented as a word cloud (better read in color - digital or print)

layer protocols like HTTP, DNS, and TLS/SSL (observed as TLSv1.2 and TLSv1.3 in the dataset) are common across most applications, while others like QUIC might be used by applications like YouTube and Spotify for improved streaming efficiency [26]. Protocols like STUN indicate real-time communication or streaming use, as seen in IP Cameras and WhatsApp. Proprietary protocols, such as TPLINK-SMARTHOME/JSON, signify specific device use, and others like EAPOL and DHCP, linked more to the network connection process, can still offer useful context. Fig. 4-A uses a circle packing algorithm to illustrate the most frequently used protocols for each application, with unique bubble colors for each application and annotations for specific protocols and their usage frequencies.

Ports. Destination ports can provide insights into the nature of the applications generating the traffic [27]. The Internet Assigned Numbers Authority (IANA) categorizes destination port numbers into three ranges: well-known ports (0-1023), registered ports (1024-49151), and dynamic/private ports (49152-65535). Well-known ports are associated with common protocols such as HTTP (port 80) and FTP (ports 20 and 21), registered ports are reserved for specific services, often used by proprietary software, and dynamic/private ports are chosen at run-time by client applications, presenting a classification challenge due to their variability across sessions. Despite the overlapping usage, especially with modern applications' preference for ports like 80 and 443, distinct patterns can often be discerned. These patterns are typically based on the frequency, timing, and correlation of destination port usage with other network features. A well-structured understanding of destination ports can significantly enhances the effectiveness of traffic classification. Fig. 4-B uses a circle packing algorithm to represent the most frequently used destination ports for each application in our dataset, with distinct bubble colors for each application and annotations indicating specific destination ports and their usage counts.

DNS Queries. DNS queries serve as an integral part of NTC, revealing discernible patterns indicative of application activities. These queries translate human-friendly domain names into IP addresses that devices use to communicate, thereby providing insights into which services or servers an application is in-

teracting with. The uniqueness and relative counts of these DNS queries can be significantly telling of an application's nature and behavior [28]. The nature, frequency, and sequence of these DNS query types can further illuminate the activity pattern of an application, aiding in the classification process. Thus, despite being a lower-level network feature, DNS queries present a highly informative dimension for network traffic classification, enabling a granular understanding of application behavior. We utilize Regular Expression (`RegExr`) [29] to extract domain names embedded within packet payloads. This method provides a mechanism to identify patterns within the chaotic text presented in the information column of the PCAP data. Regular expressions aid in discerning these patterns, effectively enabling the isolation and extraction of the requisite domain names. Fig. 5-A presents a word cloud illustrating the frequency of DNS queries associated with different applications in our dataset.

TLS/SSL Queries. Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols designed to provide secure communication over a network [30]. As part of NTC, the analysis of TLS/SSL queries found in the payload of network packets can offer valuable insights into the unique activity patterns of applications [31]. We extracted these queries from PCAP packets that use TLSv1.2, TLSv1.3, or SSL. For instance, string patterns like "Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message" are retrieved from the unstructured info column using regular expressions. These extracted TLS/SSL queries represent a series of actions performed during the secure communication process, as identified in TLS/SSL handshake protocol. For instance, "Client Key Exchange" refers to the stage where the client is establishing a secure connection with the server. "Change Cipher Spec" indicates that all future communication will be encrypted, and "Encrypted Handshake Message" represents the finalization of this secure connection. The frequency, order, and the specific type of these queries can be telling of the application's behavior and communication pattern. Thus, despite the encrypted nature of these protocols, the process flow revealed by these queries offers another level of granularity in application characterization, thereby bolstering the overall network traffic classification process. Fig. 5-B presents a word cloud illustrating the frequency of TLS/SSL queries for applications in our dataset.

VI. Multi-level Machine Learning Approach

After feature extraction from the raw PCAP in Phase-1 (see Fig. 2), we now have two types of features, *nominal* and *continuous*. The nominal parameters (ports, protocols, DNS and TLS/SSL queries) are collectively considered in the form of *bags of features* and feed into the ML-Level1 for coarse classification. The output of the ML-Level1 is input to ML-level2 along with the raw values of the continuous attributes, and their statistical mean and standard deviation values, as a 26-column feature vector (see Fig. 6). This stage delivers the fine classification output with prediction class and the associated confidence probability.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Flow Duration	Flow Length	Flow Size	Inter Arrival Time	Packet Length	Payload length	DNS predicted class	DNS prediction probability	Protocol predicted class	Protocol prediction probability	Port predicted class	Port prediction probability	TLSSL predicted class	TLSSL prediction probability	packet_length_mean	packet_length_std	payload_length_mean	payload_length_std	flow_length_mean	flow_length_std	flow_duration_mean	flow_duration_std	flow_size_mean	flow_size_std	iat_mean	iat_std
0.000444	2	142	0.465674	71	0	GoogleHome	0.999999296	GoogleHome	1	GoogleHome	1	GoogleHome	0.999953668	173.410498	166.3375736	56.15	125.946324	2.191740413	1.810992792	0.000485274	0.001182579	380.0707965	543.0147881	0.027319693	0.120427066
0	1	176	0.008907	176	110	GoogleHome	0.999999296	GoogleHome	1	GoogleHome	1	GoogleHome	0.999953668	173.410498	166.3375736	56.15	125.946324	2.191740413	1.810992792	0.000485274	0.001182579	380.0707965	543.0147881	0.027319693	0.120427066

Fig. 6. Combined 26-column feature vector (Indicating our hybrid NTC approach. Each row is a network traffic analysis entry. Each of the 26 columns is our derived feature.

Coarse Classification We realize that our Bags of Features are nominal, i.e., not considered numeric values, and are multi-valued. To elaborate, 'ports' represent a collection of destination ports used by a single application throughout the duration of packet capture. For example, when capturing packets from YouTube, we might encounter a set such as `{"50422":12801, "52800":1876}`. This set represents destination port numbers with 12,801 occurrences of port number 50422 and 1,876 occurrences of port 52800. In terms of the 'Protocols' attribute, considering Google Home packet capture as an example, we might generate a set such as `{"UDP":1090, "AJP13":324}`. This represents the protocols used, with 1,090 occurrences for the UDP protocol and 324 occurrences of the AJP13 protocol. Similar multi-valued sets arise for DNS and TLS/SSL queries. In [32] researchers have shown that A normative Bayesian model for classification (NBMC) is effective in text classification scenarios where there are many unique words, and thus we employ NBMC for coarse classification using the bag of *nominal* features. The output of this classifier comprises a class of confidence for each attribute, yielding both the predicted class and the prediction probability. During the learning phase, the classifier analyzes the distribution of words. Let us take the Bag of Protocols as an example. The classifier calculates the likelihood of each protocol name appearing given a certain class. This is computed using the formula represented in (1) [12]:

$$P(v_i|t_j) = \frac{1 + \sum_{p=1}^{I} \sum_{v=1}^{T} m_{p,t_j,v} \cdot K(v, v_i)}{T + \sum_{q=1}^{T} \sum_{p=1}^{I} m_{p,t_j,v_q}} \quad (1)$$

where, considering Bag of Protocols for Youtube as an example, v_i is the i-th unique term in the training dataset (e.g., STUN). t_i is the j-th target class (i.e. YouTube). I is the total number of instances, and T is the total number of unique terms. $m_{p,t_j,v}$ is the number of occurrences of term v in the p-th instance of target class t_j. $K(a, b)$ is the Kronecker delta function, which is 1 if $a = b$ and 0 otherwise.

During the testing phase, the classifier calculates the following probability denoted in (2) [33] for all possible target classes:

$$P(t_j|V_{\text{test}}) = P(t_j) \cdot \prod_{i=1}^{T} [P(v_i|t_j)]^{\sum_{v=1}^{T} n_{\text{test},v} \cdot K(v,v_i)} \quad (2)$$

where, $Vtest$ is a set represented by $\{v_1 : n^1_{test}, v_2 : n^2_{test}, \ldots, v_T : n^T_{test}, n^i_{test}\}$ is the occurrence number of individual unique term v_i in a given test instance, and $P(t_j)$ is the presence probability of a target class t_j in the whole training dataset (i.e., number of t_j training instances divided by total number of all training instances).

Fine Classification. As the coarse classification attributes are not linearly separable and the outputs of ML-level1 are nominal values, we use ML-Level2 to perform finer resolution classification. We employ two types of classifiers (any one is good, but include both to compare and evaluate), a random forest classifier (RFC) and a decision tree classifier (DTC). For the RFC configuration, the parameter `"n_estimators"` determines the number of decision trees used. While a larger number can enhance the model's performance, it can also increase computational demands and risk overfitting [34]. After cross-validation and tuning, we determined that three trees provided an adequate balance between computational cost and model performance. The decision to use both RFC and DTC in our system model comparison stems from several factors. Firstly, we evaluate complexity against accuracy; by comparing a DTC with an RFC, a single DTC to an RFC can reveal how increasing model complexity from one to three trees impacts accuracy. Secondly, we consider the trade-off between interpretability and performance. DTC provides superior interpretability while RFCs typically deliver better performance, thus comparison helps to optimize for transparency and efficiency. Thirdly, the comparison aids in assessing robustness against noise, outliers, and over fitting, where RFCs generally outperform due to their ensemble nature [35]. Lastly, both models yield feature-importance results, allowing us to determine the most influential features in each algorithm.

VII. Evaluations and Results

In order to assess the performance of our model and identify potential bottlenecks, we first present the end-to-end evaluation of the system performance, followed by a microbenchmark. The key results present the classifier's performance for datasets collected across multiple trials and time-windows (10 minutes, 5 minutes, 2 minutes, 90 seconds, and 60 seconds). In the coarse classification, we evaluate the performance of the NBMC in the context of Bags of Features. The fine classification examines the performance of the RFC and DTC Classifiers for end-to-end comprehensive application classification. We analyzed the quality of the model's predictions using accuracy, precision, recall, and F1-Score. In the microbenchmark, we add three applications (a total of 9 applications) in the experiment and reevaluate the entire system to test for scalability. The motivation is to examine the system's response to new applications and provide a design tutorial on how easily the system can scale. We also discuss computational complexity in the key results and microbenchmark.

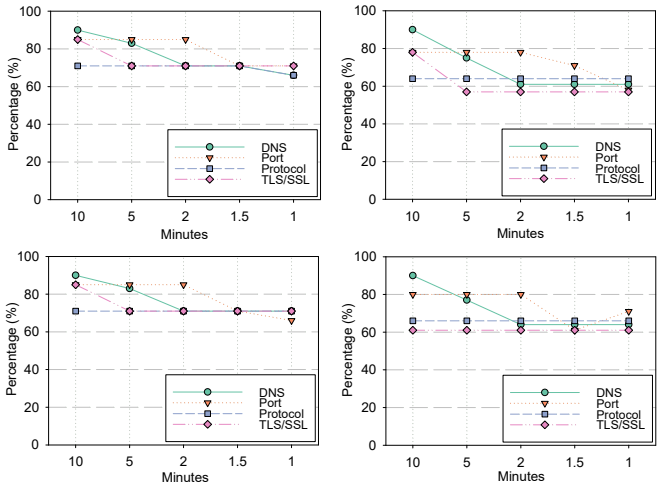


Fig. 7. Coarse Classification Performance results - Top Left Plot: Accuracy - Top Right Plot: Precision - Bottom Left Plot: Recall - Bottom Right Plot: F1-Score. X axis is PCAP data capture time window.

A. Classification Results

Coarse Classification using NBMC. The training data utilized in our evaluations are compiled from eight `'df's`. These frames are extracted from different rounds of data collection captured in PCAPs, with varied durations such as 10 minutes, 5 minutes, 2 minutes, 90 seconds, 60 seconds, 30 seconds, 20 seconds, and 10 seconds. The input data is presented in two columns, namely 'source' and 'feature'. Collectively, the training set for this evaluation is about 20 minutes and comprises 140,353 data points for the bag of protocols, 106,142 for ports, 1,989 for DNS, and 20,435 for TLS/SSL. The reason for the discrepancy in data point length stems from the intrinsic nature of these parameters. For instance, the frequency of DNS or TLS/SSL queries appearing in PCAP is notably less than that of protocols or ports. Fig. 7 illustrates the coarse classification performance of our model across four metrics: accuracy, precision, recall, and F1-score in different plots. Each line on the plot represents a different attribute: DNS (green, circle indicator), ports (orange, triangle), protocols (pink, diamond), and TLS/SSL (blue, rectangle). DNS shows the best performance at 90 percent, initially at the 10-min mark, then gradually decreasing until 2 min and stabilizing thereafter. Ports maintain around 85 percent up to the 2 min mark, then dip to 70 percent. Protocols remain stable across all intervals at 71 percent. TLS/SSL begins promisingly with 83 percent accuracy at the 10 min mark, stabilizing from the 5-min interval. However, its F1-score consistently stays around 61. It's evident that the performance breakpoint occurs at the 2-min interval, yet our four features still exhibit satisfactory results up to the 1-min mark. This indicates that capturing network traces for 1 min window our model still maintains a coarse classification performance above 60 percent. It's important to note that perfect results at ML-Level 1 are not our primary aim, however, this stage provides a foundation and probability of the coarse class identification for the next finer classification stage. That is the rationale behind proposing multi-level ML model, as the first one alone never reaches the optimum results.

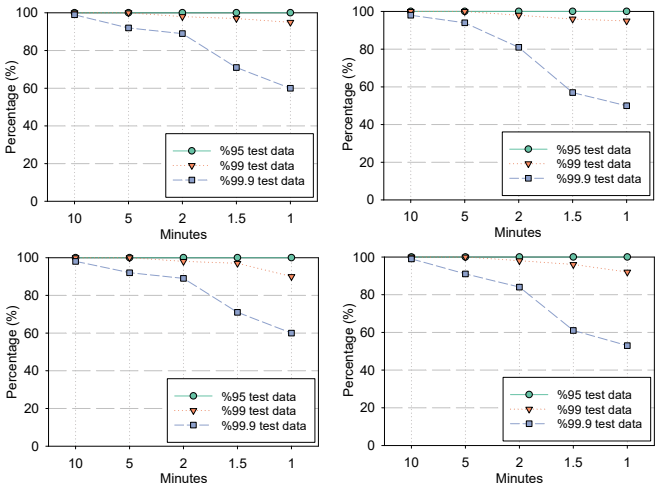


Fig. 8. Fine Classification Performance results for RFC - Top Left Plot: Accuracy - Top Right Plot: Precision - Bottom Left Plot: Recall - Bottom Right Plot: F1-Score. X axis is PCAP data capture time window.

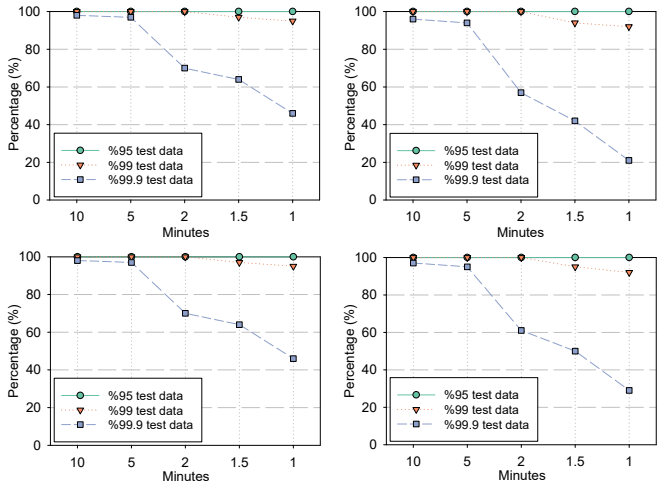


Fig. 9. Fine Classification Performance results for DTC - Top Left Plot: Accuracy - Top Right Plot: Precision - Bottom Left Plot: Recall - Bottom Right Plot: F1-Score. X axis is PCAP data capture time window.

Fine Classification using RFC. The training data for this evaluation are uniquely allocated to each trial. For instance, when testing the results for a 10-minute period, the same 10-minute dataset serves as both test and train data by varying the proportions (percentage split) of train and test data. We incorporate three variations of test:train data split: 95:5 %, 99:1 %, and 99.9:0.01%. The quantity of data points varies for each trial due to differences in the capture time windows of PCAPs. The number of data points ranges from 145,209 in a 10-minute interval to 27217 for a 120-second interval, and to 3332 for a 10-second interval. As expected, the quantity of data points diminishes as the PCAP duration decreases. For the RFC configuration, the parameter `"n_estimators"` determines the number of decision trees used. While a larger number can enhance the model's performance, it can also increase computational demands and risk overfitting. After cross-validation and tuning, we determined that three trees provided an adequate balance between computational cost and model performance. As depicted in Fig. 8, the fine classification performance of our model using RFC shows accuracy, precision, recall, and F1-score in four distinct plots. Each line on the plot represents a different test data size: 95 percent (green, circle indicator), 99 percent (orange, triangle), and 99.9 percent (pink, diamond). The results show that fine classification using RFC delivers perfect scores across all metrics for all time intervals, even at 1 min, when using 95 percent of the data for testing. With a 99 percent test data split, we retain perfect scores for the 10 and 5 min intervals, and even though there's a slight decrease from the 2 min mark, we still achieve over 90 percent accuracy at the 1 min interval. However, with a 99.9:0.01 % split, satisfactory results are only seen at the 10 min interval, with a significant decrease for the 1.5 and 1 min intervals. In summary, using a 95 percent test data split assures perfect classification. However, we also can observe that our system can sustain upto 90% accuracy at 99:1 % data split, even with 1 min network traces capture. **This means that it is possible to execute the classification of applications using our system in real-time within a 1 minute time buffer window.**

Fine classification using DTC. Fig. 9 reveals that fine classification with DTC exhibits similar trends with both 95 and 99 percent test splits. With a 99.9 percent split, results mirror RFC for the first two intervals, but then drop sharply to approximately 60 percent at the 2-min interval, further declining to around 40 percent for accuracy and recall, and 20 percent for precision and F1-score in the last two intervals. Overall, DTC matches RFC's performance at the 10 and 5-min intervals, even surpassing RFC at the 2-min interval with a 99 percent test split. However, performance drops significantly beyond this point. DTC remains a valid option considering a 95 or 99 percent test data split.

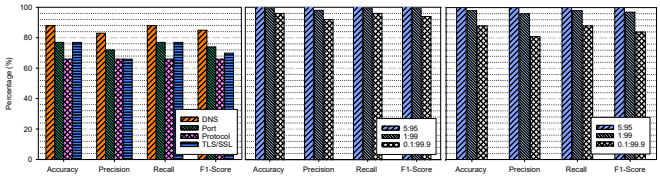


Fig. 10. Mircrobenchmark for Scalability - Left Plot: Coarse Classification - Middle Plot: Fine Classification RFC - Right Plot: Fine Classification DTC

B. Microbenchmark

Scalability test. We add three more applications to our experiments and collect PCAP data considering those as a part of our networking experiment environment. Additional training data (considering the new applications) for coarse classification is gathered from a 30-minute capture, yielding 212,716 data points for a bag of protocols, 162,833 for ports, 2,638 for DNS, and 30,797 for TLS/SSL. Fig. 10 presents system performance for coarse and fine classifications for a 5-minute evaluation interval. Observing Fig. 10 (left), trends for DNS, port, protocol, and TLS/SSL mirror those from our key results, albeit with a minor decrease (around 5 percent) for each metric. Despite this, results remain satisfactory, all exceeding 60 percent, and confirming that the system performance remains consistent at Phase-1 even when more applications are added. In Fig. 10

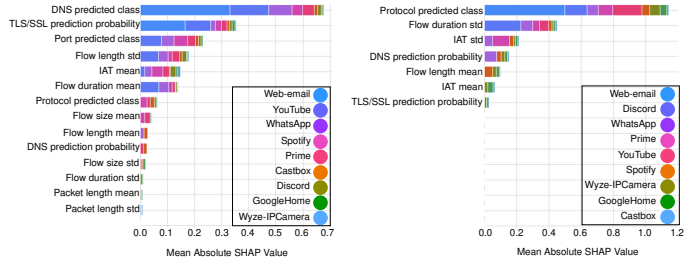


Fig. 11. Microbenchmark - SHAP feature importance for RFC (left) and DTC (right). Each color corresponds to a specific application. For instance, in RFC (left) figure, the feature "DNS predicted class" primarily contributes to the classification of "web-mail", followed by "YouTube", and then "WhatsApp", among others.

(center), compared to the fine classification using RFC, reveals only minor variations except for the system performance with a 95 percent test data split, which is perfect. Using a 99 percent test data split gives us a near-perfect performance of 98 percent, compared to the prior perfect score. This slight decrease can be considered almost negligible. The behavior of the 99.9 percent test data size parallels that in the key results, with a minor reduction (less than 5 percent). In Fig. 10 (right), concerning fine classification using DTC, reveals a consistent trend only for the 95 percent test data split, showing perfect system performance. However, the 99 percent test data split performance lags slightly behind RFC and there's a significant drop when considering the 99.9 percent test data split, with precision dipping to about 80 percent, which isn't satisfactory. In summary of the scalability test, our results with nine applications are fairly consistent with that of six applications, demonstrating system stability and scalability. The system remains robust with a 95 percent test data split, yielding perfect classification confidence.

Feature importance. Fig. 11 shows the SHAP (SHapley Additive exPlanations) feature importance when using RFC vs DTC for our fine classification. SHAP attributes the contribution (importance) of each feature to the prediction for each sample [36]. The displayed feature importance underscores the effectiveness of our proposed hybrid NTC and multi-layered ML system for fine classification. The plot reveals the influence of both nominal and continuous features. Our 'Bags of Features' approach, applied in coarse classification, has significantly aided in accurate classification, as demonstrated by importance of features such as "DNS predicted class", "TLS/SSL prediction probability", "Protocol predicted class", etc. Furthermore, statistical distribution from packet-level, payload-level, and flow-level (analytical features), as seen in "packet length mean", "payload length std", "flow size std", "flow length mean", "IAT std" (interarrival time), also contribute significantly. We infer that the distribution of the flow and packet features were more helpful than their raw values.

VIII. Conclusion

In this paper, we introduced and assessed a novel application classification system using a software-defined wireless network experimentation approach. Our four-phased system architecture seamlessly integrated a hybrid network traffic classification approach with a multi-level machine learning method. Our analysis uncovered distinctive patterns and behaviors for each application, particularly emphasizing nominal parameters like ports, protocols, DNS, and TLS/SSL queries. Feature selection from a diverse set of packet-based, payload-based, statistical-based, and behavioral-based attributes, resulted in successful classification and provided insights into feature importance. Our extensive evaluation demonstrated exceptional performance for both coarse and fine classifications across various PCAP data capture volumes and time windows. Notably, our model achieved near-perfect application classification accuracy with just a 1-minute data capture, utilizing only 5 percent of the data for training. System scalability was confirmed with the addition of three more applications, and key features influencing classification were highlighted through SHAP analysis. As future work, we plan to assess the system's performance through a microbenchmark focused on in-app service identification, addressing the unique challenges posed by applications like WhatsApp. While acknowledging our contribution isn't the first to apply ML models to network traces, our approach offers a distinctive perspective on the significance of each network feature in application classification. This underscores the importance of selecting the right tools and features, advocating for thoughtful problem-solving rather than blindly following trends.

IX. Acknowledgements

This work was supported by the National Science Foundation(1901133) and was supported in part by UDSA-NIFA(2021-67019-34337).

References

[1] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[2] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *International workshop on passive and active network measurement*, pages 41–54. Springer, 2005.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[4] Md Ariful Haque and Dr Rajesh Palit. A review on deep neural network for computer network traffic classification. *arXiv preprint arXiv:2205.10830*, 2022.

[5] Muhammad Shafiq, Xiangzhan Yu, Asif Ali Laghari, Lu Yao, Nabin Kumar Karn, and Foudil Abdessamia. Network traffic classification techniques and comparative analysis using machine learning algorithms. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 2451–2455. IEEE, 2016.

[6] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, 16(2):1135–1156, 2013.

[7] Hamid Tahaei, Firdaus Afifi, Adeleh Asemi, Faiz Zaki, and Nor Badrul Anuar. The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538, 2020.

[8] Ola Salman, Imad H Elhajj, Ayman Kayssi, and Ali Chehab. A review on machine learning–based approaches for internet traffic classification. *Annals of Telecommunications*, 75:673–710, 2020.

[9] Openwrt project. https://openwrt.org/. Online; accessed 11-May-2023.

[10] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of iot devices. In *Proceedings of the 2018 workshop on attacks and solutions in hardware security*, pages 41–50, 2018.

[11] Ahmed Saeed and Mario Kolberg. Towards optimizing wlans power saving: Novel context-aware network traffic classification based on a machine learning approach. *IEEE Access*, 7:3122–3135, 2018.

[12] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2018.

[13] Elaiyasuriyan Ganesan, I-Shyan Hwang, Andrew Tanny Liem, and Mohammad Syuhaimi Ab-Rahman. Sdn-enabled fiwi-iot smart environment network traffic classification using supervised ml models. In *Photonics*, volume 8, page 201. MDPI, 2021.

[14] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE access*, 5:18042–18050, 2017.

[15] Tal Shapira and Yuval Shavitt. Flowpic: A generic representation for encrypted traffic classification and applications identification. *IEEE Transactions on Network and Service Management*, 18(2):1218–1232, 2021.

[16] Miguel Camelo, Paola Soto, and Steven Latré. A general approach for traffic classification in wireless networks using deep learning. *IEEE Transactions on Network and Service Management*, 2021.

[17] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. A review on random forest: An ensemble classifier. In *International conference on intelligent data communication technologies and internet of things (ICICI) 2018*, pages 758–763. Springer, 2019.

[18] Priyanka and Dharmender Kumar. Decision tree classifier: a detailed survey. *International Journal of Information and Decision Sciences*, 12(3):246–269, 2020.

[19] Salvatore Costanzo, Laura Galluccio, Giacomo Morabito, and Sergio Palazzo. Software defined wireless networks: Unbridling sdns. In *2012 European Workshop on Software Defined Networking*, pages 1–6. IEEE, 2012.

[20] Dual GbE Carrier Board with 4GB RAM, 32GB eMMC RPi CM4 Case. https://www.seeedstudio.com/Dual-GbE-Carrier-Board-with-4GB-RAM-32GB-eMMC-RPi-CM4-Case-p-5029.html, Year of access (e.g., 2023). Online; accessed 11-July-2023.

[21] Luci: A web-based graphical user interface for openwrt. OpenWrt Project, Year of publication.

[22] tcpdump - a powerful command-line packet analyzer. Available online at https://www.tcpdump.org/, Latest version accessed in 2021.

[23] Wireshark Foundation. Wireshark: The world's foremost and widely-used network protocol analyzer. Available: https://www.wireshark.org. Accessed: accessed 19-June-2023.

[24] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. Mimetic: Mobile encrypted traffic classification using multimodal deep learning. *Computer networks*, 165:106944, 2019.

[25] May Thae Naing, Thiri Thitsar Khaing, and Aung Htein Maw. Evaluation of tcp and udp traffic over software-defined networking. In *2019 International Conference on Advanced Information Technologies (ICAIT)*, pages 7–12. IEEE, 2019.

[26] Armir Bujari, Claudio E Palazzi, Giacomo Quadrio, and Daniele Ronzani. Emerging interactive applications over quic. In *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–4. IEEE, 2020.

[27] Klenilmar Lopes Dias, Mateus Almeida Pongelupe, Walmir Matos Caminhas, and Luciano de Errico. An innovative approach for real-time network traffic classification. *Computer networks*, 158:143–157, 2019.

[28] Minzhao Lyu, Hassan Habibi Gharakheili, Craig Russell, and Vijay Sivaraman. Analyzing enterprise dns traffic to classify assets and track cyber-health. *arXiv preprint arXiv:2201.07352*, 2022.

[29] G Skinner and team. RegExr: Learn, Build, & Test RegEx, 2021. accessed 3-June-2023.

[30] Norbert Pohlmann. Transport layer security (tls)/secure socket layer (ssl). In *Cyber-Sicherheit: Das Lehrbuch für Konzepte, Prinzipien, Mechanismen, Architekturen und Eigenschaften von Cyber-Sicherheitssystemen in der Digitalisierung*, pages 439–473. Springer, 2022.

[31] Jan Luxemburk and Tomáš Čejka. Fine-grained tls services classification with reject option. *Computer Networks*, 220:109467, 2023.

[32] Gurinder Singh, Bhawna Kumar, Loveleen Gaur, and Akriti Tyagi. Comparison between multinomial and bernoulli naïve bayes for text classification. In *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, pages 593–596. IEEE, 2019.

[33] Laurent Valentin Jospin, Hamid Laga, Farid Boussaid, Wray Buntine, and Mohammed Bennamoun. Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, 2022.

[34] Gavin Smith, Roberto Mansilla, and James Goulding. Model class reliance for random forests. *Advances in Neural Information Processing Systems*, 33:22305–22315, 2020.

[35] Jan Brabec and Lukás Machlica. Decision-forest voting scheme for classification of rare classes in network intrusion detection. *CoRR*, abs/2107.11862, 2021.

[36] Yasunobu Nohara, Koutarou Matsumoto, Hidehisa Soejima, and Naoki Nakashima. Explanation of machine learning models using improved shapley additive explanation. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 546–546, 2019.