Communication Networks - Final Project

Contributors: Melissa, Aviad, Daniel

Part 1:

1. There can be a few factors in the transport layer that contribute to slow data transfer. One of them is congestion control. TCP uses congestion control algorithms to detect congestion in the network; when congestion is detected, TCP reduces packet transmission rates. Another factor that contributes to slow data transfer is high packet loss, which causes TCP to retransmit lost packets. The most efficient way to troubleshoot is to analyze the sent packets and received packets to determine packet loss and retransmitted packets.

2. TCP's flow control ensures reliability between the sender and the receiver. It guarantees that all the packets have been sent and received. The flow control ensures that the sender is sending at a speed at which the receiver can handle without packet loss.

   If the sender has significantly higher processing power than the receiver, TCP flow control will slow down the speed at which the user can send packets to the receiver.

3. When there are multiple paths between the source and the destination, the path choice affects the network by how much traffic is being passed through the same route. A factor that should be considered in routing decisions is balancing traffic between the different routes. If the "fastest" path is always chosen, it can create an infinite path-switching cycle where the pathfinder constantly switches back and forth between two paths.
4. MPTCP improves network performance by distributing our data through multiple network interfaces, which has a larger bandwidth than a single TCP connection. This causes a more reliable connection because if one route fails, then it can be switched to another route. MPTCP also dynamically changes traffic routes based on network conditions, switching when a faster route is found.
5. At the transport level, one of the causes of packet loss can be congestion in the network. If one router is sending packets at a speed too great for the other router to handle, then the receiver will have to discard packets it doesn't have place to store in its buffer. A step we can take to solve this is to check if packets are only being routed through certain paths and try to reconfigure the network so they flow through more or different routes.

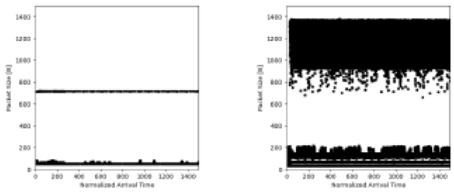
6. At the network layer, a reason for packet loss can be network failure. We can use tools like ping to check the network connection. Packet loss can also occur when there are routing issues, and we can use traceroute to pinpoint where the packet loss occurs.

Part 2:

Paper 1 Summary

"FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition"

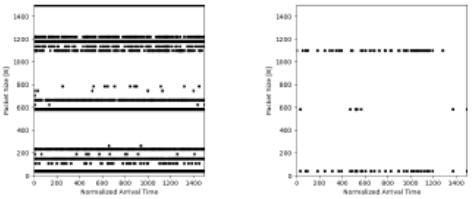
In the paper, the authors introduce a novel way to identify and classify internet traffic. The paper's main contribution, is via converting a network traffic "Flow" into an Image, and applying classical Image Recognition techniques – specifically Machine Learning – to reasonably classify the traffic.



(a) Netflix (b) Skype

Figure 1: Examples of FlowPics for Netflix and Skype video applications. Note that for illustration purposes, black pixels represent any value between 1 and 255, while white pixels represent the value 0 (namely, there is no arrival of packets with the corresponding size in the corresponding delta time.).

Such an approach is novel in the way it recognizes traffic, since it relies on novel Machine Learning systems. It does so, by creating "FlowPics" from traffic flows (sliced into time-windows) then graphing the received packet size over time – creating a chart image to be fed into CNN models for training and later recognizing. Taking advantage of time and size related features makes approach is generically applicable – even to encrypted traffic, as it does not rely on traffic content thus maintaining privacy. It also makes use of surprisingly small amounts of data for such accuracy, making it a viable real-time-classification option.

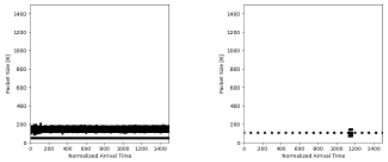


(e) Tor: VoIP (f) Tor: Chat

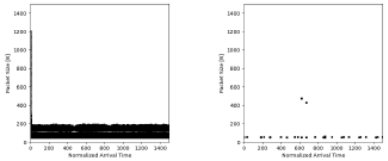
Figure 2: Examples of FlowPics, where the left column corresponds to VoIP traffic, the right column corresponds to chat traffic, and each row corresponds to a different encryption technique.

The main results from the paper were highly impressive, with a remarkable near 99.9% average correct classification accuracy.

Problem	FlowPic Acc. (%)	Best Previous Result	Remark
Non-VPN Traffic Categorization	85.0	84.0 % Pr., Gil *et al.* [15]	Different categories. [15] used unbalanced dataset
VPN Traffic Categorization	98.4	98.6 % Acc., Wang *et al.* [7]	[7] Classify raw packets data. Not including browsing category
Tor Traffic Categorization	67.8	84.3 % Pr., Gil *et al.* [15]	Different categories. [15] used unbalanced dataset
Non-VPN Class vs. All	97.0 (Average)	No previous results	
VPN Class vs. All	99.7 (Average)	No previous results	
Tor Class vs. All	85.7 (Average)	No previous results	
Encryption Techniques	88.4	99. % Acc., Wang *et al.* [7]	[7] Classify raw packets data, not including Tor category
Applications Identification	99.7	93.9 % Acc., Yamansavascilar *et al.* [10]	Different classes



(a) Non-VPN: VoIP (b) Non-VPN: Chat



(c) VPN: VoIP (d) VPN: Chat

Paper 2 Summary

"Early Traffic Classification With Encrypted ClientHello: A Multi-Country Study"

The paper presents a solution to the challenging problem of traffic classification, while dealing with encryption, specifically the Encrypted ClientHello (ECH) extension to TLS. The paper's main contribution is a new hybrid algorithm, hRFTC (hybrid Random Forest Traffic Classifier), which leverages both the remaining unencrypted TLS metadata and novel flow-based features, along with a large and diverse multi-country dataset.

The method is new because it combines several ideas. It updates an older way of looking at packet data (RB-RF) so it works with both regular encrypted traffic (TLS-over-TCP) and newer traffic (QUIC/HTTP3). More importantly, it uses new information about the flow of traffic, like packet sizes and timings. It's unique because it only looks at packets up to the point where the server first sends real data. This is fast enough for real-time use. Because it doesn't look inside packets, it keeps user data private.

TABLE 10. Comparison of the packet-based classifiers on the subsets of the dataset.

Selected Traffic Subset	Avg TLS Distinction of the Traffic Subset	Macro F-score [%]		
		RB-RF	BGRUA	MATEC
Live Video	1	100.0	100.0	100.0
Short Buffered Video	0.99	99.2	81.5	82.3
Buffered Video	0.87	87.4	75.7	75.9
Buffered Audio	0.71	71.8	59.9	63.0
Vk	0.65	66.2	57.1	61.1
Google	0.58	55.5	55.0	54.7
Facebook	0.53	52.6	51.4	51.2
Yandex	0.49	40.2	39.8	38.1

The paper's results are very strong. The new hRFTC method is much better than existing methods at correctly identifying traffic types, even when those methods use similar techniques. The results show that looking at traffic flow patterns is very important, especially when strong encryption (ECH) is used. While hRFTC works well even with limited training, it's crucial to train it with data from the *same location* where it will be used. The system is also fast, processing 100,000 connections every second.

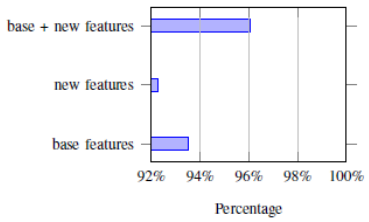
Paper 3 Summary

Analyzing HTTPS Encrypted Traffic to Identify User's Operating System, Browser, and Application

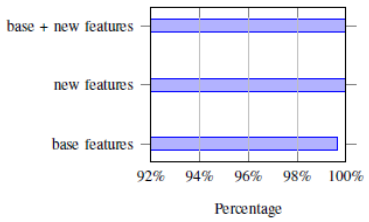
This paper introduces a novel approach to identifying a user's operating system, browser, and application by passively eavesdropping on HTTPS traffic. The authors achieve this through supervised machine learning techniques.

The authors have extracted and utilized new features that are based on SSL behavior, TCP features, and the bursty behavior of browsers. It then uses a combination of traditional traffic features (packet sizes, inter-arrival times, etc.) and the new features to improve classification accuracy.

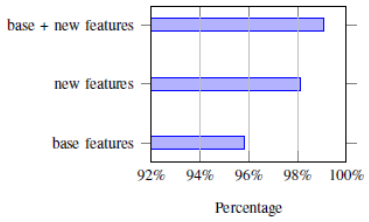
The main results are a high classification accuracy of the user's operating system, browser, and application when using supervised machine learning techniques and a combined set of base and new features. In addition, we see that the new features significantly improved classification accuracy compared to using only the base features. The accuracy improved from 93.51% to 96.06%.



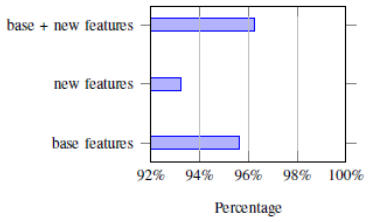
(a) Tuple Accuracy Results



(b) OS Accuracy Results



(c) Browser Accuracy Results



(d) Application Accuracy Results

Part 3:

Summery:

Our analysis compares the traffic patterns of different applications: YouTube, Discord, Gmail (Firefox), Chrome, and Spotify. We use the scapy and matplotlib libraries plot our extracted data. Our data includes features such as packet size, inter-arrival time, TCP flags, TCP window size over time, as well as the protocol distribution. By extracting and charting such data, we can then compare and classify each session. We present our charted data below.

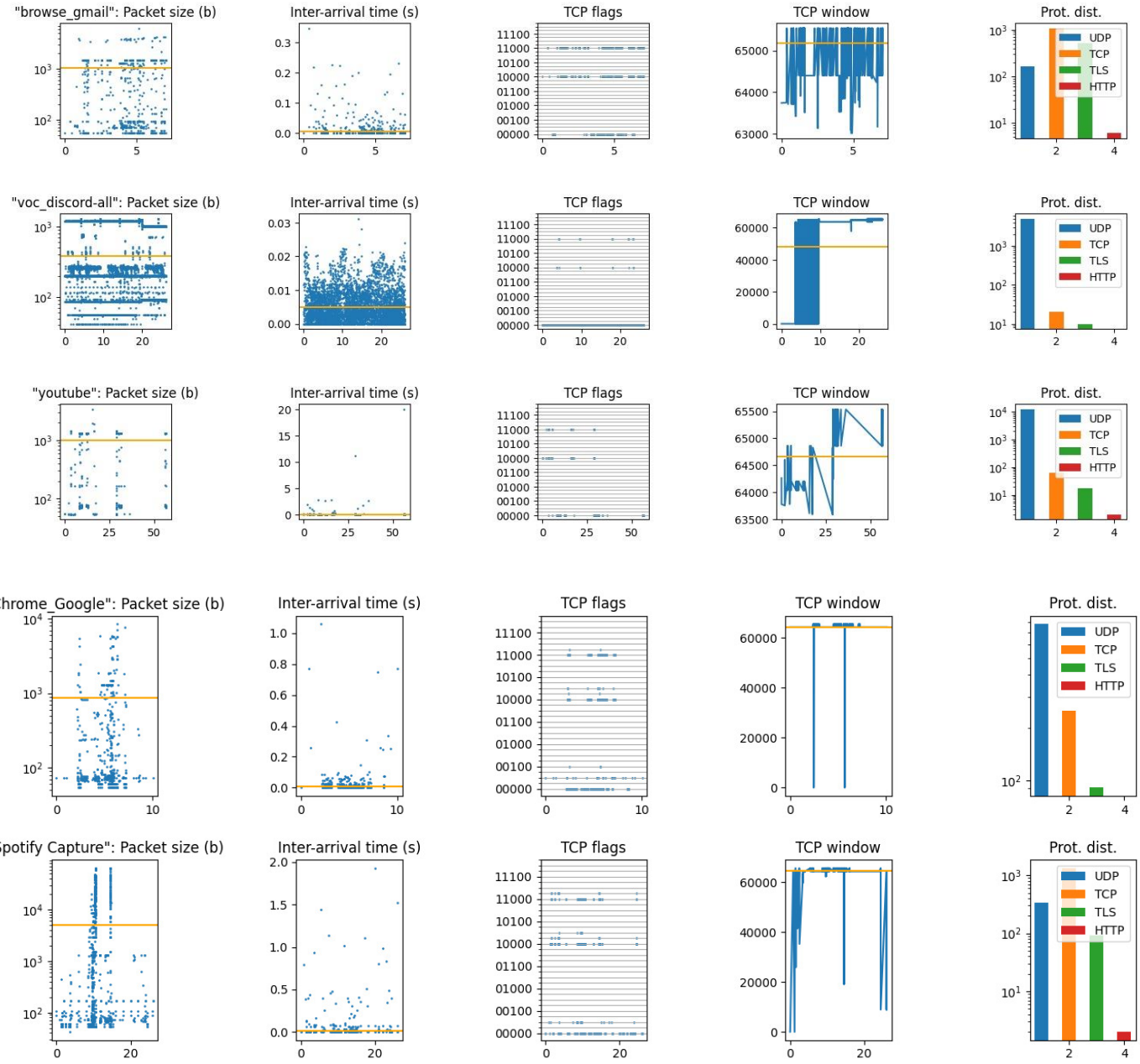


Chart observations, analysis & insights:

We observe in our charts that video streaming applications such as YouTube, seem to use mostly UDP packets, as they prefers fast data transfer over reliable connection. We further observe that YouTube typically has larger packet sizes than the other applications we recorded because it transfers high quality video chunks which typically take more data. YouTube also shows long inter-arrival times, sometimes reaching up to 20 seconds. This suggests that video content is downloaded in bursts rather than continuously transmitted, which reflects YouTube's video buffering functionality. YouTube also shows a wide range of TCP window sizes, indicating dynamic congestion control.

For VoC applications such as Discord, which is a live video and audio streaming platform, we observe mostly UDP packets as well, as it too prefers fast data transfer over reliable connection. Discord displays a mix of small and large packet sizes with distinct clutters. This pattern aligns with real-time voice and video communication, which involves smaller packets for audio and larger packets for video streams. Discord has very short inter-arrival times, indicating frequent and rapid packet exchange, which is crucial during live video and audio streaming. This platform maintains a relatively stable TCP window size, indicating a continues connection with little downtime.

Gmail is an application used on a web browser, in this case on the Firefox browser. In our observations we see it relies more on TCP and TLS, with less UDP. This is expected, as web browsing and email services prioritize data reliability and secure connections over speed. Gmail has a large variety of packet sizes, but many are relatively small. This aligns with web surfing, where requests and responses are generally smaller (loading text, images, and emails). Gmail has low inter-arrival times, which suggests a steady but less frequent data exchange, consistent with web browsing, where users request pages intermittently. Gmail, like YouTube, shows a wide range of TCP window sizes, indicating dynamic congestion control as the downtime between user actions allows the window size to adjust.

Chrome is a web browser that primarily relies on TCP and TLS for secure web browsing. Like Firefox, Chrome displays a wide range of packet sizes including many small packets and some larger ones. This pattern is once again expected for web browsing, where small packets handle requests, and larger packets handle data transfers. We see short inter-arrival times with occasional spikes, suggesting bursts of traffic as web pages load and respond dynamically to user actions. Chrome has a high but stable TCP window size, indicating efficient congestion control and bandwidth management for web traffic.

For audio only streaming, we've tracked the packets sent by Spotify, an audio streaming application. We observe that Spotify uses mostly TCP and TLS with some UDP packets, which suggests it prioritizes reliable data delivery over speed. This tracks with Spotify's priorities to play accurate and reliable sound – as

opposed to call applications such as Discord, which care less about audio quality. Spotify shows a mix of packet sizes but has noticeably larger packets than browsing, which aligns with media streaming behavior, where music data packets are transferred. We also see fluctuations in TCP window size, which could indicate adjustments for buffering or varying network conditions while streaming.

Spotify has the biggest packet size out of all the applications we analyzed, sometimes exceeding 10^4 bytes. Then YouTube follows, typically using larger packets. Discord has a mixed range of packet sizes, as well as Chrome. Gmail has the smallest packet size compared to the rest.

Discord has very short inter-arrival times, with many more data points concentrated near zero, meaning that packets arrive very frequently. Gmail also has short inter-arrival times with larger gaps than Discord. Chrome has short inter-arrival times with some spikes. Spotify and YouTube, on the other hand, have larger inter-arrival times compared to the other applications.

All the applications show a distribution of TCP flags with various bit patterns, indicating different control and data packets, which we can use as further data to classify the traffic.

Discord and Chrome have the most constant TCP window size as opposed to Spotify and YouTube, which show a large variation of TCP window sizes. Gmail's window size is relatively steady compared to Spotify and YouTube.

YouTube and Discord use primarily UDP packets, as opposed to Gmail and Chrome, which use mostly TCP and TLS with fewer UDP packets. Unlike YouTube and Discord, which mainly rely on UDP for real-time content, Spotify uses primarily TCP and TLS.

Recognizing traffic (as an Attacker):

In a scenario where an attacker knows each packet size, time stamp, and flow ID, we show that by observing the charts created for a given Flow an attacker can easily match traffic flows to known services. Websites or applications that most people use on a day-to-day basis often communicate with specific servers (e.g., Google); therefore, the attacker can relate specific Flow IDs to certain websites or servers by analyzing datasets of common traffic patterns.

In a scenario where an attacker only knows each packet size and time stamp, the features extracted by our charts still show sufficient data such the attacker can identify what applications has been used. Different applications create different patterns of packet exchange. An attacker can compare these patterns against a database of common traffic patterns. In addition, different types of services have distinct timing patterns

that can also be compared against a database of known patterns. An attacker can also identify websites that have been accessed by analyzing inter-arrival delays. We thus show that, Even if the data is encrypted, the time intervals between packets can provide telling information. An attacker can recognize websites based on packet sizes and packet bursts.

A potential solution to such an attack would be for applications to try and hide the true size and arrival time of data, by perhaps adding random overhead to each packet to hide their true size and sending junk-like fake packets to conflate the arrival data. Such an approach however will come at an obvious logistical cost on both ends of the application and potentially increase the stress on the network as a whole – therefore any similar solution must weigh such costs against potential benefit of such an obfuscation strategy.