Soon Gi (Andrew) Shin
Iman Nematollahi
Stephen Doan
Shrimant Singh
Samson Qian

Res Recovery: Classifying Video Resolutions Through a VPN Tunnel

Abstract

Virtual private networks, or VPNs, have seen a growth in popularity as more of the general population has come to realize the importance of maintaining data privacy and security while browsing the Internet. In previous works, our domain developed robust classifiers that could identify when a user was streaming video. As an extension, our group has developed a Random Forest model that determines the resolution at the time of video streaming with 87% accuracy.

Introduction

VPNs reroute their users' network traffic data through their own third-party private servers, allowing them to disguise the precise details of their users' network activity and provide consumers complete anonymity. Though the usage of online security applications is certainly beneficial to its consumers, VPNs prove to be an issue for Internet Service Providers (or ISPs) who lose valuable information about their customers' network traffic data that could be used to improve the efficacy of their services. In particular, this loss of detail creates difficulties for ISPs when trying to properly understand their customers' needs and troubleshoot their users' network connectivity issues. For example, customers may experience long buffering times when browsing the web but without knowing the exact circumstances regarding their activity, ISPs are unable to effectively provide helpful solutions to remedy their users' issues.

Over the course of ten weeks, students mentored by the Viasat community were able to construct robust classifiers that are capable of determining whether or not a user is streaming video while connected to a VPN. Our project is designed to be an extension of this past work as we develop a multi-class classification model that can identify the resolution of a video that was viewed over a VPN connection. The assumption for our project is that the classifier that we build will serve as the second half of a two-part pipeline, where the inputted data is first categorized as being video or non-video, and, when appropriate, is then provided an additional label specifying resolution.

The premise of our project initially seemed relatively simple as, intuitively, we believed that streaming video at higher resolutions would require more bandwidth. This assumption was true but only to a certain extent. Depending on the user's network conditions and capabilities, the

thresholds separating the 6 different resolutions would often begin to overlap between users. Application of spectral analysis (Fourier transform) helped to alleviate some of this overlap but it still proved to be difficult to construct a model that is able to precisely identify the numeric resolution value for a given video. Therefore, our final classifier opts for a "low", "medium", and "high" labeling system that serves to provide a general but concise range of resolutions that a user might potentially be streaming video at.[1] Given this tool, ISPs will be able to notice when their users are consistently streaming video at much lower resolutions than they should be capable of and help them to resolve such problems.

Methods

Data

We had two primary requirements when selecting a video streaming provider: full autonomy over video quality selection and the ability for a single video to be streamed at multiple resolutions. For these reasons, we decided to use YouTube as our primary source..

Network traffic data was recorded using Viasat's script, Network Stats.[2] The script allows us to observe the transactions of data that occur between two unique IP addresses each second. Since video streaming data at any resolution utilizes a larger magnitude of bandwidth than local network activity, further data cleaning to isolate VPN activity was not necessary. To supplement the Viasat script, we developed a Selenium script to help automate the data collection process. Keeping our previous data requirements in mind, the script was designed in such a way that the user only needs to provide a YouTube playlist containing all of the desired videos as well as a list of target resolutions. Once this information is inputted, the script then streams each video at each of the given resolutions and collects network traffic data for each video/resolution pair. For our data set, each group member captured data for a single Youtube video six times, once for each of the six resolutions that we had selected beforehand. This was key to our data collection process as it ensured consistency in the sizes of the training data collected for each label and it minimized the influence of any potential external factors. Assigning each member the task of collecting data over their own network also helped to incorporate a variety of different network conditions into our model, improving its overall robustness. Our final dataset is composed of data collected from 20 unique videos at the resolutions 144p, 240p, 360p, 480p, 720p, and 1080p for a total of over 25 hours worth.

Since Network Stats collects observations by the second, grouping the data into chunks (a section of continuous data) is necessary to generate sufficient useful information for engineering features. A larger chunk size is preferred due to the fact that our classifier will be less prone to misclassifying sudden influxes of bytes. However, for the purposes of our project, a smaller chunk size is also preferred since it would allow users of our tool to be able to quickly capture and classify their data. We determined that 5-minute chunks were the best length for describing

[1] High = 720p & 1080p; Medium = 360p & 480p; Low = 144p & 240p

[2] Laubach, Charles (2020) "network-stats", https://github.com/Viasat/network-stats

network behavior. This 5-minute data overhead means that users of our model will have to collect more data but the increased accuracy is a worthy tradeoff. As such, all of our network traffic data captures were preprocessed to be 5-minute samples. Moving forward, it would be correct to assume that we are working exclusively with 5-minute chunks and that all features were created from 5-minute chunks.

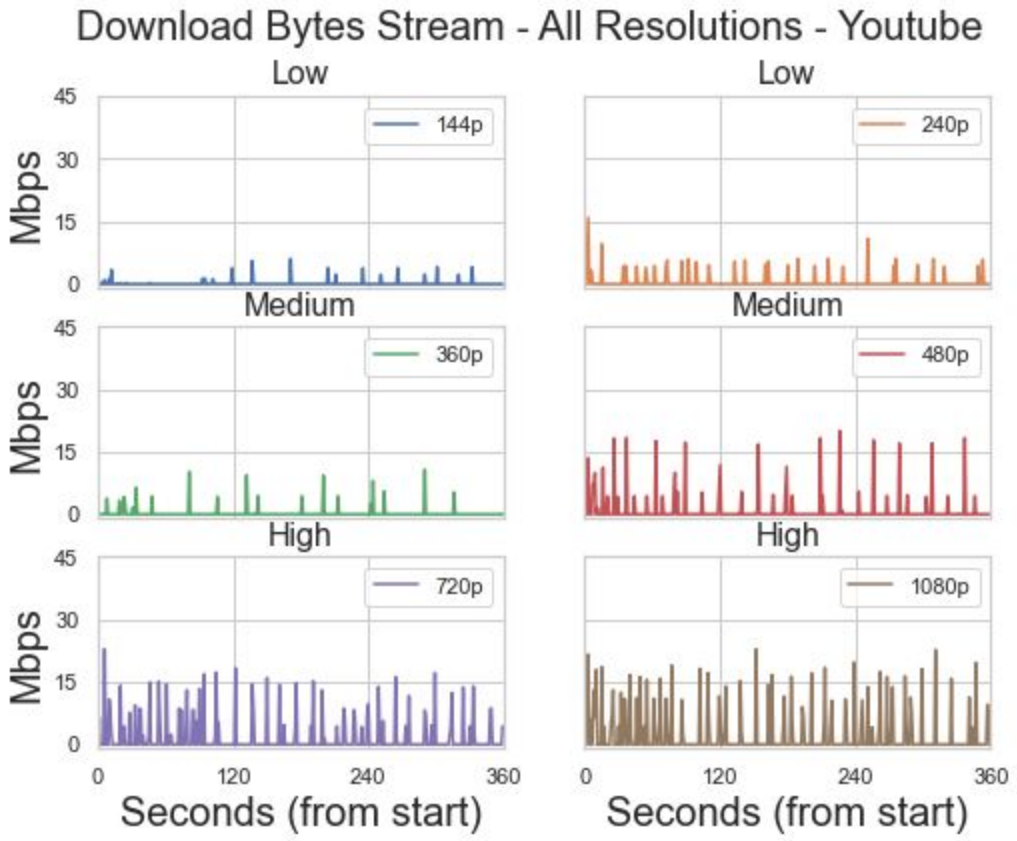
Features

Our features can be categorized into 3 distinct categories: basic aggregate/thresholding features, spike analysis-derived features, and spectral analysis-derived features (in the frequency domain). The basic aggregate/thresholding features were designed to establish some preliminary boundaries that could separate the resolutions based on observations we made during our exploratory data analysis. Our spike analysis also yielded additional features that helped to define some of the unique characteristics of each resolution. And, given the data's signal-like properties, we also leveraged spectral analysis to engineer features that could identify the frequency of bytes being sent from the VPN server. Note that our group made the conscious decision to focus primarily on the download direction of bytes as opposed to the upload.[3] Preliminary EDA showed that behaviors between the two byte stream directions are mirrored - download is simply larger in magnitude of bytes. If necessary, a full table of the features can be found in the appendix.

1. *Analyzing Byte Stream*

Our initial features came from analyzing the download bytes stream. As stated previously, we know that streaming at higher video qualities generally requires more data to be sent over the network in order for the given video to be viewed properly. Based on our analysis, we were able to conclude the following: captures for high resolution videos had the highest frequency of data being transmitted and the largest magnitude of bytes. Captures for medium resolution videos exhibited a similar magnitude of bytes but at a lower frequency. Captures for low resolution videos had both the smallest magnitude of bytes as well as the lowest frequency of bytes sent. With these results in mind, we decided to use the average and standard deviation of bytes per second as features. The average and standard deviation values create basic thresholds that give a general idea of how much data is required for each resolution. Below is a distribution of the downloaded bytes per second for a single 6-minute video captured at every resolution. We can visually see that as the resolution increases, the average number of bytes being downloaded increases as well.

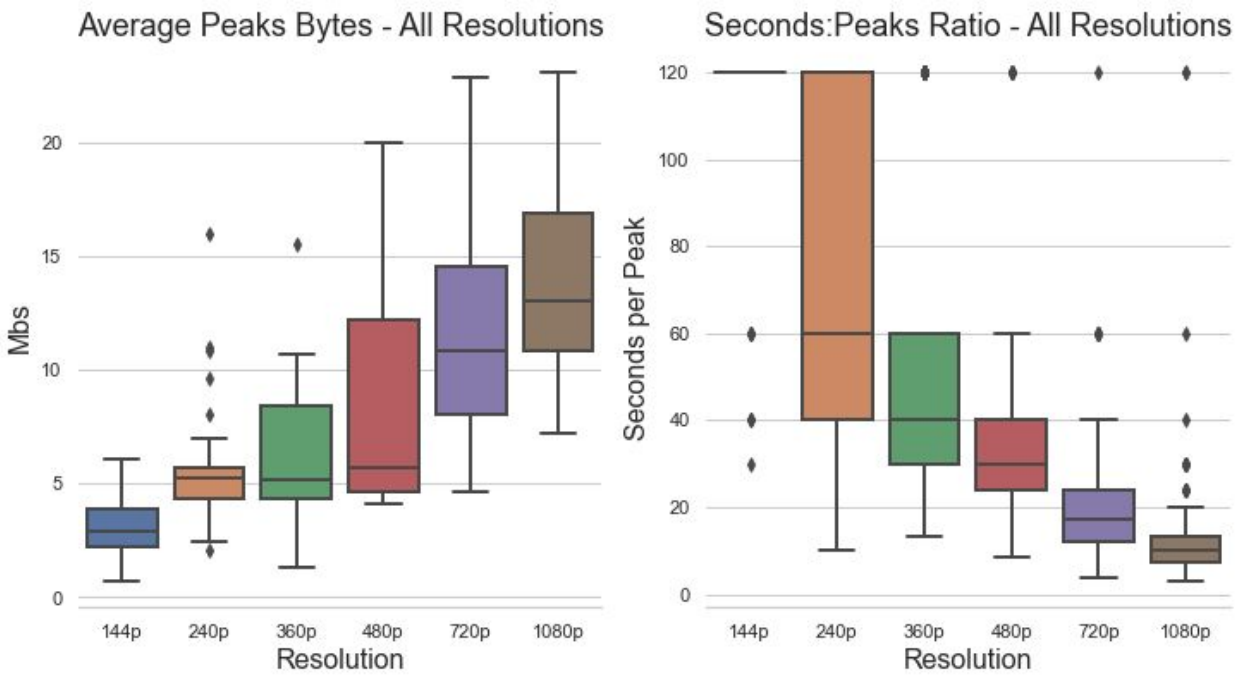
[3] download: VPN server to local machine; upload: local machine to VPN server



2. *Data Peaks/Spikes*

Data peaks/spikes are defined as being 'large' transactions of data that are received over the network in a given second. Finding a minimum threshold value for determining whether or not a data point should be considered a peak took some experimentation but we ultimately decided upon 5 megabits (Mbs) as our minimum value.[4] Using this, we were able to extract a total of 4 features: the average and standard deviation of the peak byte size, the total number of peaks, and the seconds-to-peak ratio. The most interesting feature that we extracted was the ratio of seconds to peaks. Originally, we tried taking the average of the time intervals (in seconds) between peaks to use as a feature instead, however, due to outliers in the data, we discovered that it was not as discriminative as we had formerly believed. In particular, this feature failed to be useful in cases where the network would send large amounts of data in rapid succession for a low-resolution video capture. Because this trend tends to happen only in captures for high-resolution videos, our model would often misclassify these low-resolution video captures as being high-resolution. As such, we developed the seconds-to-peak ratio as a feature since it is robust to this corner case of a sudden peak influx.

[4] 5 megabits = 625000 bytes



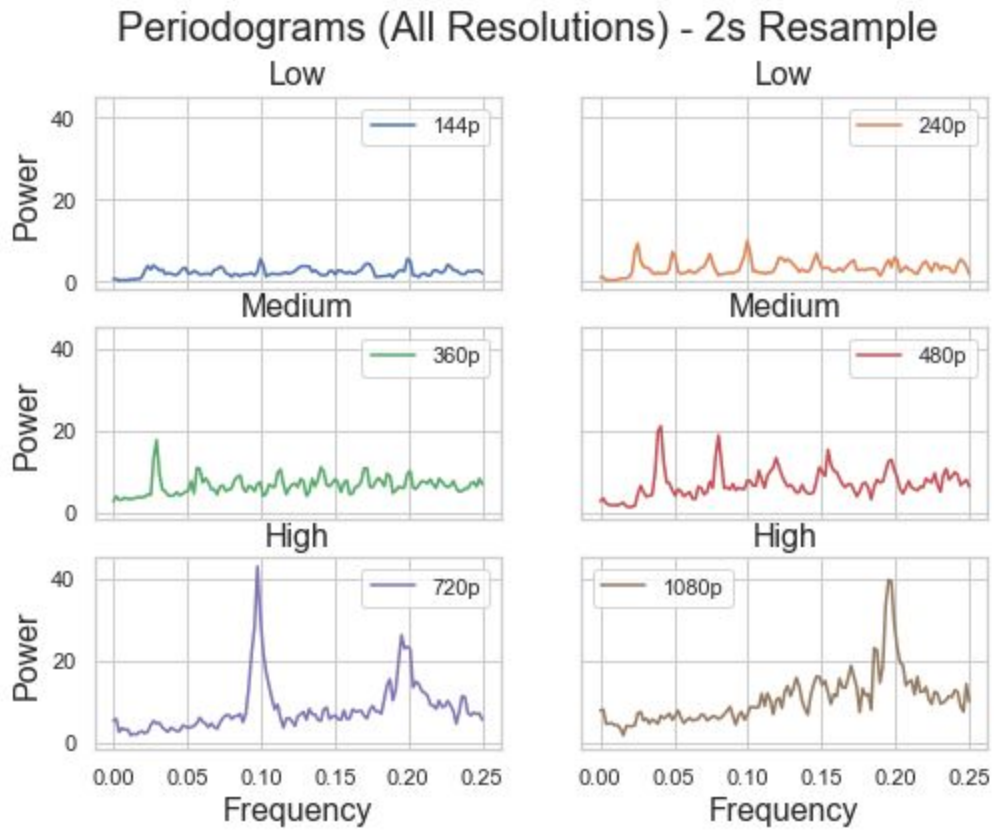
The plot on the left illustrates how, in general, as the resolution increases, the average peak byte size increases as well. On the right, we can see that the ratio of seconds-to-peak decreases as the resolution increases, which illustrates how the quantity and consistency of peaks increase as the quality improves.

3. *Spectral Analysis*

After extracting some features from within the time domain, we shifted our focus to the frequency domain. We did this by first resampling our data into 2-second intervals and then utilizing an application of the Fourier Transform called Welch's method. When applying Welch's method on clean data and plotting the periodogram, you should expect to see a well-defined peak that is significantly larger than the rest, signifying which frequencies are the strongest in your data. In our analysis, we found that these well-defined peaks are present in the periodograms for high-resolution video data but are less apparent in the periodograms for low- or even medium-resolution video data. For the captures of the low and medium qualities, we saw multiple peaks of similar magnitudes, but no single peak that stood out.

In terms of feature engineering, we identified the peaks (no hard threshold; relative to the chunk) shown in the periodograms and calculated their prominences. Prominence describes how much larger in magnitude a peak is in relation to its neighbors. We saw that captures for high-resolution videos consistently produced large maximum prominence values, captures for medium-resolution videos produced lesser maximum prominence values, and captures for low-resolution videos produced the smallest maximum prominence values. Thus, the first spectral analysis-derived feature that we generated was the maximum peak prominence. To capture the fact that the spread of the peaks in captures for lower resolution videos is smaller than its higher counterpart, we took the standard deviation of the maximum prominence values

as a feature as well. In the frequency domain, the bandwidth requirements to support streaming high resolution video results in signals whose power values are much greater than their lower resolution counterparts. We show this down below where we also plot the frequency distributions and have scaled the y-axis to be the same across all graphs to highlight these differences.



Results

For our classifier, we ultimately decided to implement a Random Forest model. Since our features are fairly correlated with each other, we hoped that the inherent randomness of the Random Forest algorithm would overcome some of this collinearity. The algorithm was also selected because it allows us to see which features are heavily influential in the decision process and which features fail to make an impact.

Baseline Model

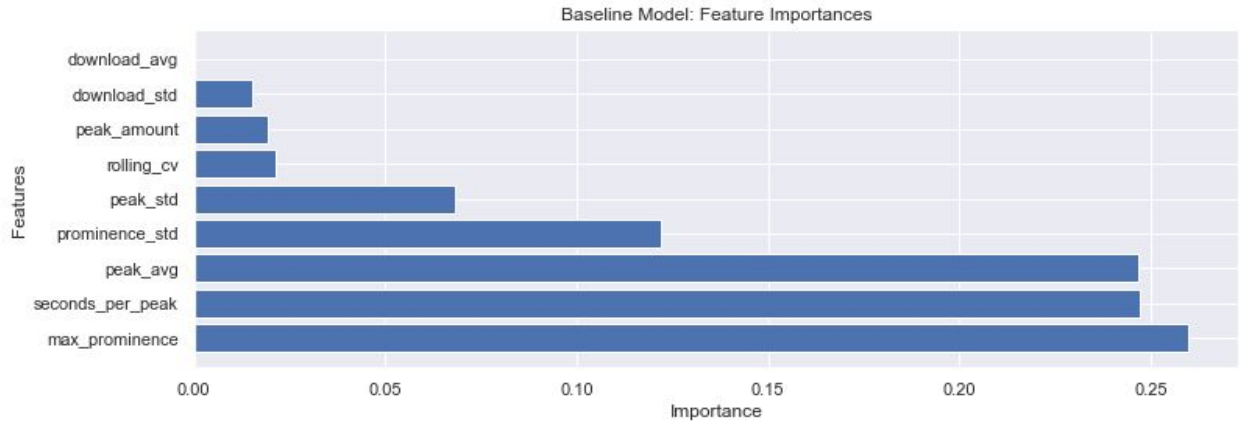
Our baseline model was made to ensure that the features we had created and extracted were significant enough to differentiate resolutions that were farther apart. We took a subset of our training data (240p, 480p, 1080p), trained a Random Forest classifier, and achieved an accuracy of 91% with the 9 aforementioned features. Below is the normalized confusion matrix

and F1-scores for each label. We can see that the model performs perfectly for 240p, but tends to struggle slightly for the higher resolutions. The classifier also does not misclassify 240p as 1080p, and vice versa. However, we do see that 480p is sometimes mistaken as being either 240p or 1080p.

Prediction Resolution / **Actual Resolution**	240p	480p	1080p
240p	1.00 (18)	0 (0)	0 (0)
480p	0 (0)	0.85 (17)	0.15 (3)
1080p	0 (0)	0.11 (2)	0.89 (17)

F1 Score

240p	480p	1080p
100%	87%	87%



We can see above that the features that are closely related to the amount of downloaded bytes [“peak_avg”, “prominence_std”, & “max_prominence”] tend to have higher importances than the other features used in the classifier. This is to be expected as we saw that there are considerable differences in the bandwidth requirements for the 3 selected resolutions.

Extended Model

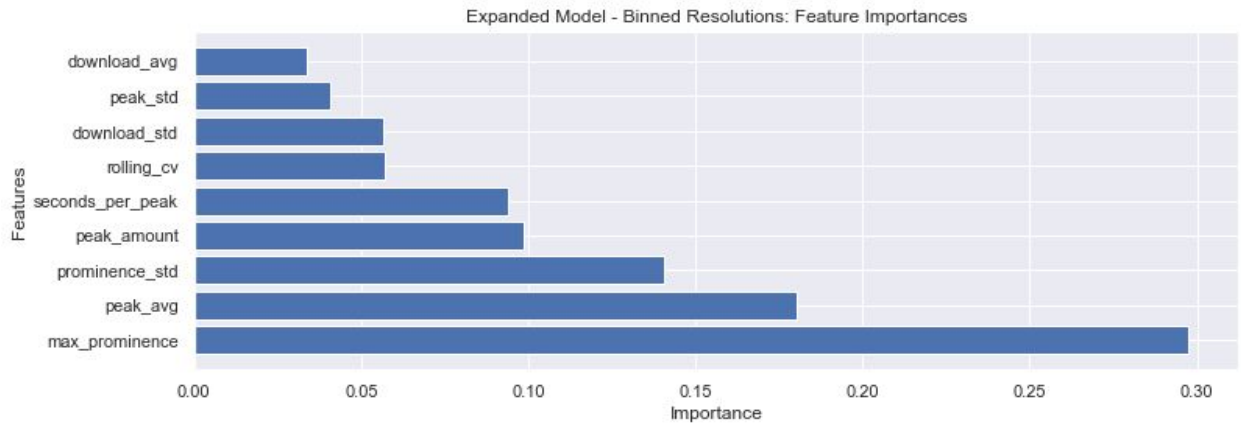
Originally, we planned to expand our model to be able to predict the exact resolution. However, once we utilized the full training set, our accuracy dropped drastically to 73%. Trying to create a model that could accurately label the numeric resolution value for quality proved difficult so we opted to bin the resolutions. Our final model takes in a network traffic data

capture and outputs either “low”, “medium”, or “high” as the label. For the purposes of our project, low corresponds to the resolutions 144p & 240p, medium corresponds to 360p & 480p, and high corresponds to 720p & 1080p. Using the same 9 features as before, our final model is able to produce an accuracy of 87%.

Prediction Resolution / **Actual Resolution**	Low	Medium	High
Low	0.88 (29)	0.12 (4)	0 (0)
Medium	0.08 (3)	0.84 (33)	0.08 (3)
High	0.02 (1)	0.08 (3)	0.90 (35)

F1 Score

Low	Medium	High
88%	84%	91%



As stated previously, our classifier experienced a decrease in overall accuracy. We can also see that the F1-score for the ‘medium’ label is the lowest out of the 3. This suggests that there are instances within captures for medium-resolution data that closely resemble low- or high-resolution data. In addition, we see that the standard deviation of the peak byte sizes within a chunk was not a good differentiator of resolutions; this could be due to the fact that the spikes in a chunk do not vary drastically in byte size, resulting in similar standard deviation values across the whole dataset.

We can also see that a 2-class jump misclassification has occurred, where a high-resolution sample of data is predicted as being low-resolution. This exemplifies one of the

potential issues with selecting a smaller chunk size to utilize. Observing the data, there are many scenarios where a small subsection of a high-resolution data can sometimes resemble low-resolution data or even non-video data. For example, when Youtube plays an ad, the server stops sending data during the duration of the ad and the level of network activity greatly decreases. Increasing the chunk size would likely help aid this problem, however, the 5-minute data overhead is quite intense already and, in our opinion, increasing this requirement is not worth the small increase in accuracy.

Discussion

Given the overall success that we had in creating our models, we decided to try to take our project a step further and attempted to create a more extensive model that would be capable of classifying the exact resolution, and not just the degree of quality. After visualizing various plots for each resolution, we were able to visually identify differences amongst the 6 classes, however, we were unsuccessful in generating useful features that implemented these differences. We fed numerous different combinations of our features; however, we were faced with issues in accurately classifying 720p. The model performed well for all of the other resolutions, but would often misclassify an overwhelming amount of 720p captures as being 1080p.

Looking towards the future, one approach that could be taken to extend our work would be to engineer additional discriminative features that could help to establish a more accurate model that is able to classify specific resolutions. Another approach could be to include a larger variety of resolutions into our classifier such as 1440p and higher. Other interesting topics might include an analysis of the amount of information that can be extracted from network traffic data. Analyzing the differences in the network traffic data for popular and unpopular YouTube videos or distinguishing the content type of a video (such as action vs. still) are topics that could be of interest as well.

Overall, we found the extent of the information that we were able to learn from the network traffic data that we collected through a VPN connection to be fascinating. However, while working with this data, discussion over the amount of knowledge that should be able to be extracted while someone is using a VPN was common amongst our members. VPNs are supposed to establish a sense of privacy for the user, and while these tools are created with the intention of ultimately fostering a better experience for the customer, it is important to keep ethical implications such as this in mind.

Appendix

Laubach, Charles (2020) “network-stats”, https://github.com/Viasat/network-stats

Feature Definitions:

Features	Description
download_avg	The average bytes per second in the download direction.
download_std	The standard deviation of the bytes in the download direction.
peak_avg	The average byte amount of data peaks in the download direction. Peaks are defined as any single second transfer of data that is greater than 5 megabits.
peak_std	The standard deviation of byte amount of data peaks in the download direction.
peak_amount	The total number of data peaks in the download direction.
seconds_per_peak	The ratio of seconds to peaks.
max_prominence	Max prominence of the strongest signal in a periodogram (represented as the tallest peak in the graph). The frequency and power values are generated by applying Welch’s methods on a 2 second resampled version of the data. Peaks in the periodogram are found by using SciPy’s find_peaks method and the prominence values are calculated for data points considered peaks.
prominence_std	In the periodograms, high resolution data generates well defined peaks while lower resolution does not. In our max_prominence feature, we calculate the prominence values for all the peaks found in the data. We simply run take the standard deviation of this array to create this prominence_std feature.
rolling_cv	Coefficient of variation taken over a rolling window version of the data