Virtual Private Network Detector (VPND)

Serhat Kiyak
Carnegie Mellon University
skiyak@cs.cmu.edu

Chase Miller
Carnegie Mellon University
cnmiller@andrew.cmu.edu

Nicholas Caron
Carnegie Mellon University
ncaron@andrew.cmu.edu

***Abstract*—Through VPNs, any user in the world can access some content that is only available to people in a particular country and restricted for the rest. This may violate regulations about copyrights, broadcasting rights, and also cause loss of profit and waste of resources for the content provider. Therefore, there is a motivation to detect VPN servers and block them. In this paper, we explore ways to detect VPN traffic by analyzing HTTP headers, TCP traffic patterns, and geolocation information. We present the Virtual Private Network Detector (VPND) tool that detects VPN traffic with high true positive and low false positive rates. We evaluate our tool with real world experiments using various browsers and VPN extensions.**

I. Introduction

With the increasing popularity of content providers that broadcast TV shows, movies, and sporting events; a legal concern has been raised that VPN users can access content that is restricted in their countries. This content includes anything that has copyright issues together with the companies' commercial decisions or governmental requirements to restrict some content in various parts of the world. For example, Netflix only exists in the market of a set of countries whereas it doesn't provide content to a majority of countries. However, a user that lives in a country where Netflix does not exist in the market, can easily VPN through a country where Netflix broadcasts content. Another case is related to sporting events that are subject to restrictions in the viewer's country. For instance, ESPN had broadcasting rights for the 2014 FIFA World Cup games in the United States whereas ZDF had these rights in Germany. These rights make it illegal for these channels to broadcast to other countries.

There are several concerns with VPN tunnels enabling users all around the world access the content broadcasted in a specific country. First of all, it causes a waste of resources for the content provider, and this may result in a lower quality of service for legitimate users. Secondly, there are legal issues forcing the provider to restrict the content within some set of countries, therefore, there may be legal issues behind broadcasting to other countries through VPN tunneling. Finally, the provider can choose not to broadcast some content in a set of countries for commercial reasons, and VPN tunneling would violate commercial policies. For all these reasons, a content provider would want to detect such VPN connections and block them when necessary.

In this paper, we explore ways to detect VPN traffic on the server side by analyzing the incoming traffic. Our results show that there are certain behaviors of VPN connections that provide good heuristics for detection. We come up with three mechanisms to detect VPN traffic: analyzing HTTP header fields, TCP traffic patterns and geolocation information. We design and implement a tool, Virtual Private Network Detector (VPND) that detects VPN traffic with high true positive and low false positive rates. VPND is publicly available at: https://github.com/serhatkiyak/VPND.

II. Related Work

Most existing VPN traffic detection methods implemented by content providers rely on a predetermined blacklist of IP addresses that are known to be associated with VPN providers. Incoming connections are referenced against this blacklist, and if they are known to be associated with a VPN service, the content which the VPN client was attempting to access is restricted.

In this paper, we strive to take VPN detection a step further by performing real time traffic analysis on known VPN traffic. In order to effectively analyze this traffic, we rely on preexisting research in the area. First, the Berger et al. [1] provides an overview of the existing VPN technologies. This paper serves as a general reference when identifying VPNs.

Yildirim et al. [3] provides a number of approaches for classifying IP traffic with encrypted data flows. They aim to identify encrypted, tunneled traffic in such a way that firewalls and network optimizers can make use of that decision. This is important because it is likely how a content provider, such as Netflix, would implement a similar solution to detect VPN traffic.

Wright et al. [2] provides examples of an advanced strategy for reducing the effectiveness of traffic classifiers on encrypted traffic. Being familiar with the traffic morphing techniques that are introduced in this paper and are implemented by some VPN services, provides us with patterns to look for in our traffic analysis.

Most related work is not directly targeted at VPN traffic detection, and to the best of our knowledge this paper presents the first focused research on VPN traffic detection. Regardless, the common feature among all approaches is that there is a danger of false positives. In this paper, we aim to design a tool that detects VPN traffic with no false positives. Otherwise, we believe content providers would be unwilling to leverage such a solution that endangers them to block legitimate users.

III. Methodology

In this paper, we focus on server-side VPN traffic detection. Because we do not have access to the content providers'

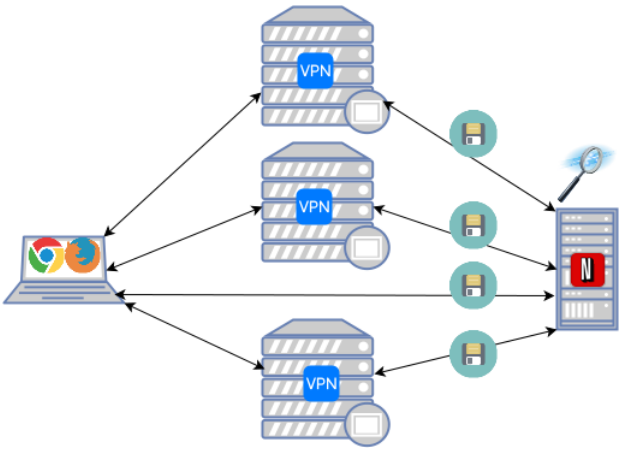

Fig. 1. VPND Methodology

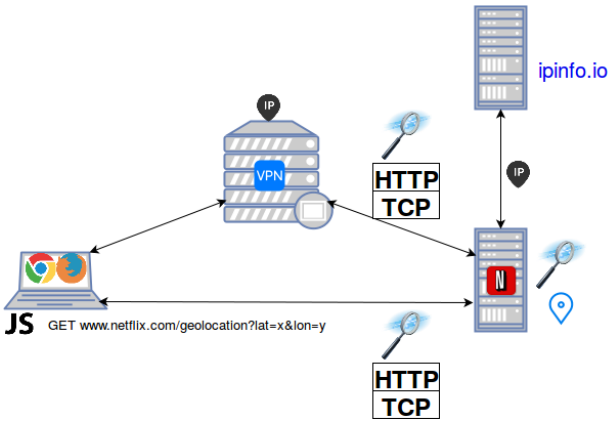

Fig. 2. VPND Design

production servers, we setup Apache servers on Amazon EC2 instances that serves various types of content. We use free website templates to mimic realistic websites that are commonly used and these templates have content similar to real world websites such as HTML, CSS, JavaScript files and images. We also copied the web content served by Netflix as their index page, and hosted the page on our server to experiment with.

For analyzing HTTP headers and TCP traffic patterns, it is sufficient to use unmodified page contents. However, for geolocation analysis as we present in Section IV, we modified the JavaScript code to be able to obtain the geolocation information from the true client.

The second component of our experiment setup is the browsers we used. We particularly picked Mozilla Firefox and Google Chrome as the browsers we experiment with. The main reason is that they are the most popular browsers that would provide a good coverage for realistic web traffic and also they come with numerous available VPN extensions that we choose to use in our experiments. We prefer to use VPNs that are browser extensions, because we anticipate most clients, such as Netflix users, probably tend to use these extensions because they are simple and convenient to use.

The third component is the VPN extensions we used. Particularly we experimented with Browsec, DotVpn, GomVpn, Hola VPN and Tunnel Bear VPN as Chrome extensions; and Hoxx, and KProxy as Firefox extensions.

We experimented with a client sending requests to our server both with and without a VPN. By capturing all incoming HTTP traffic, we had the opportunity to observe packet headers and any VPN signatures within them.

Figure 1 pictures the methodology for the experiment setup. It displays the server which hosts a copy of the Netflix index page. The browsers on the client side send HTTP requests to our server to retrieve the page. It connects to the server without any VPN and with various VPN proxies to enable the server to collect the data. The server records all incoming HTTP packets for manual inspection to find VPN specific signatures on the packets. Finally, we inspect the HTTP traffic and find ways to detect VPN traffic. Note that even though this process is not real time, VPND does real time detection to enable the server to block IP addresses that belong to VPN proxies as soon as possible.

Finally, we implemented the VPND tool based on the VPN signatures we identified. We implemented the tool in Python code and deployed it in our experiment server. The tool detects the VPN traffic and displays the source IP address that belongs to the VPN proxy. This information is important if the content provider is building a blacklist of known VPN proxy IP addresses.

IV. Tool Design

Fig. 2 pictures the high level design of VPND. VPND, which runs on the server, analyzes incoming HTTP and TCP packets in real time. First of all, it marks the connection as VPN traffic, if it finds any VPN signature in the HTTP header fields. Secondly, it generates a TCP traffic pattern as observed, and compares this pattern to the expected pattern for a connection with the specific Round Trip Time (RTT) to the connected machine. If the two patterns differ significantly, the tool marks the connection as VPN traffic. Finally, it compares the geolocation of the IP address the server is talking to with the geolocation of the true client and if they don't match, it raises a VPN alert. We explain the design of the three detection mechanisms in the following subsections in detail.

A. Geolocation

We believe that there is an opportunity to detect VPN traffic by exploiting the distance between the VPN proxy and the client. In this paper, we are mostly attacking VPN connections that are from various countries. This is to simulate users attempting to access content that is restricted in their own countries. Therefore, we believe this distance would be significant, indicating a VPN connection on the server side. VPND has two separate ways to learn the location of the client and the VPN proxy, which we describe in detail.

1) Client geolocation: VPND gets the geolocation information from the client by JavaScript execution that generates a URL with the user's embedded coordinates, and sends a

request to the server. Since, the proxies do not modify the request URLs, the server receives this information correctly. The following URL pictures the geolocation URL with random coordinates:

http://www.VPNDserver.com/geolocation?lat=39.35&lon=33.78

Note that the server doesn't even need to host a file named geolocation, since this URL is only consumed by VPND tool. We wrote 14 lines of JavaScript code to enable this functionality, and it is a pretty straighforward technique.

2) VPN proxy geolocation: VPND gets the geolocation information based on the sender IP address by contacting a publicly available information service at ipinfo.io. This service replies with a bunch of information including the hostname, network, city, address, and geolocation coordinates. Then, VPND extracts geolocation coordinates from the response.

3) Comparison of geolocations: Once VPND has two sets of geolocation information, it measures the distance between the two in terms of miles. If the numbers are unexpectedly high, VPND concludes that the IP address the server is talking to is a VPN proxy.

One may ask why not just find the country from the geolocation of the client, and then block traffic from that client if the service is not available to that country. We haven't chosen this for two reasons. First, it introduces a scalability issue, since VPND now needs to examine all connections without any exception. With our design, sampling would reduce the amount of traffic examined, and it would still result in successful detection of VPN traffic. Once VPND decides a connection is coming from a VPN proxy, it can block that address and doesn't need to examine any traffic from that IP address anymore. Second, with such a design there is no way to block VPN traffic from a client, if the client does not consent to share its geolocation information.

B. HTTP Headers

Analysis of HTTP request headers gives us another opportunity to detect VPN traffic. During our initial manual packet inspection, we identified that each of the VPN services we were surveying modified HTTP headers in its own, unique way. This header modification came in two primary forms: HTTP header ordering and the introduction of a unique identifying HTTP header.

1) Header Field Ordering: The most distinguishing characteristic of VPN HTTP headers is their ordering. As mentioned in Section III, we focused our analysis on the traffic of two browsers, Mozilla Firefox and Google Chrome. Each of these browsers has a standard HTTP header ordering which we first identified. Then, we repeated the manual packet inspection for the VPN traffic from each of the VPN browser extensions that we were testing, recording the ordering of each of their respective HTTP headers. The results of this analysis can be found in Table I.

Looking at Table I, it is easy to see that each of the VPN browser extensions that we investigated had its own means

TABLE I
HTTP HEADER FIELD ORDERING

	Chrome						Firefox		
Header Field	No Proxy	Browsec	DotVPN	GomVPN	Hola	Tunnel Bear	No Proxy	Hoxx	Kproxy
Host	1	1	1	1	1	1	1	1	1
Connection	2	10	10		2	10	7	7	6
Pragma	3	2	2	8		7			
Cache-Control	4	9	9	5	10	5		6	
Accept	5	3	3	2	4	3	3	3	3
User-Agent	6	4	4	9	3	2	2	2	2
DNT	7	5	5	6	5	6			
Referer	8	6	6	7	8	8	6		5
Accept-Encoding	9	7	7	3	6	9	5	5	
Accept-Language	10	8	8	4	7	4	4	4	4
X-Hola-Version					9				

of ordering the HTTP header fields. In addition to simply reordering the fields, some VPN servers omitted standard header fields altogether.

To implement the HTTP header field ordering check in VPND, we first created lists containing the standard HTTP header field ordering for our browsers. Then, as VPND is ingesting traffic, it collects the incoming HTTP header fields into a temporary list. This list is then compared against our standard ordering list. If an HTTP header field is detected out of order, an alert is raised.

2) Unique Headers: Another method for detecting VPN traffic via HTTP headers is to look for the introduction of unique, identifying HTTP headers that are appended by the VPN server. In the VPN browser extensions that we surveyed, the only unique HTTP header that we identified was *X-Hola-Version* which was appended by the Hola VPN proxy. Other common HTTP headers, which we didnt encounter but are sometimes added by VPN services, include *X-Forwarded-For* and *Forwarded*.

VPND is designed to accept a blacklist of these known VPN-associated HTTP header fields. With the blacklist populated, VPND then has the ability to analyze traffic in real time, searching for these known VPN HTTP header fields, and raise an alert if one of them are detected. Although it is becoming less common for VPN providers to append these unique fields, this is an easy way to detect and filter some of the more obvious VPN traffic.

Although using HTTP headers by themselves may not be enough to definitively differentiate between normal and VPN traffic, its inclusion in VPND introduces yet another layer of identification to accurately characterize traffic.

C. TCP Traffic Pattern

A third methodology, though not as necessarily thorough as the others, would be in the form of TCP traffic pattern analysis. This acts as a nice tie-in to geolocation, as it again, is an approximation of location, but this time, we use network level indicators to attempt to indicate where proxied user may lie. In particular, we aim to battle against two differing metrics, the observed round trip time to the proxy, and the actual round trip time to the true client.

Our reasoning follows some very simple logic, all of which can be analyzed during the TCP handshake. We first can wait for a connection to come in from a user, requesting that a TCP connection be initiated. Should we ping this host from our end, and the target host is using a browser VPN, we should be able

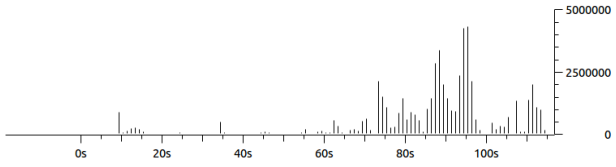

Fig. 3. Throughput With VPN Enabled from US to Japan

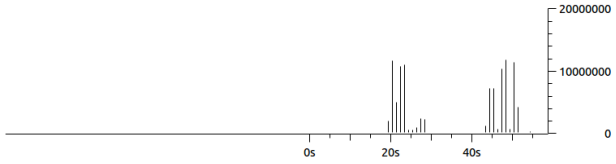

Fig. 4. Normal Throughput from US to Japan

to send a select ping, such as via ICMP, that will result in a direct response that avoids the VPN. We then can estimate the VPN induced RTT, by starting recording from the moment we send our SYNACK back to the user. We then record the time again at the responding ACK, and can now compare the two round trip times. Since all our analysis is done within a generally small period, we can aleviate some fears of the effects of an unstable network, as our measurements are made within the scope of our TCP handshake, a very small time period, which allows for very little chance for major upsets.

One might wonder whether the VPNs in question actually induce such a large amount of latency, but in reality this is very much the case for the use cases we wish to investigate. In particular, we are often are looking at cross-reigon users, who will be attempting to access a piece of content from a large enough physical distance, where a non-optimal path may induce noticible latency. In fact, we saw via measuring bitrate differences between VPN usage requests, and general requests, that bitrate is in fact halved on average. We can see this though the graphs below, in which we have two measures of bitrate in bits per second. Fig. 3 shows our bitrate for VPN enabled traffic, whereas Fig. 4 shows the bitrate for normal traffic, painting this picture in a far easier to see light. Again, the bitrate for the VPN enabled traffic is half that of the latter, making this an interesting metric to pursue.

This is a noticible difference, and allows us to therefore define a fair threshold for measurement: if our perceived real RTT to a host is half that of the estimated RTT through the TCP handshake. With this in mind, we can have a nice additional metric to measure when looking for VPNs, that has the added convenience of being low impact from fair users, and calculable just as a connection is established.

Of course, this does not come without some drawbacks. A more robust VPN client may be able to wise up to our ways, and also route our ping traffic through itself, or an unfair client may purposefully induce latency at this step to stymie this process. In addition, since cross-regional traffic is a big contributor to our investigation in this metric, some missed detection may occur for more closely routed VPN usage. However, this will certainly catch some impure users,

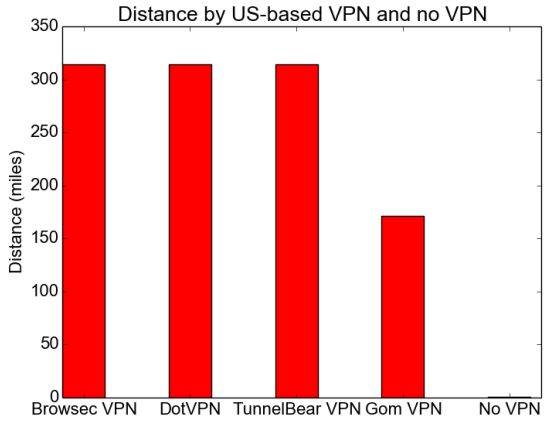

Fig. 5. Distance to US based VPN proxies

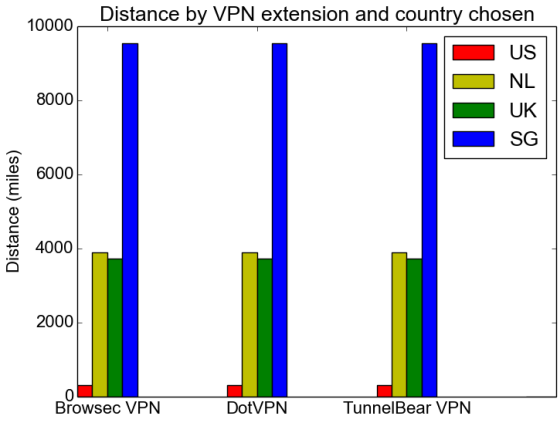

Fig. 6. Distance to VPN proxies residing in various countries

and provides a good additional metric to continue exploring alongside our others.

V. Evaluation

VPND's goal is to detect VPN traffic with high accuracy, considering and eliminating all cases that are regular traffic. We evaluate all three mechanisms of VPND separately.

A. Geolocation

There are some challenges that we need to consider due to obtaining the geolocation information from the client through JavaScript execution:

- **Browser support:** We find that all major browsers support geolocation services even with earlier versions. Particularly, Google Chrome, Microsoft Edge, Mozilla Firefox, Safari, and Opera support it. Therefore, we believe such an implementation would have high coverage.
- **Privacy:** We understand sharing geolocation information is voluntary, therefore the browser would show a popup to ask for the permission of the user.
  First of all, we need to think of what happens if the client doesn't share this information. The good thing in this

approach is that even a small portion of clients sharing the information would be sufficient enough to detect and block VPN proxies given the server would still have some distances to refer.

Second, websites may be unwilling to ask for this information due to privacy policies. Note that there are some websites that already ask the client for the geolocation information for some functionality, and for these websites our approach is trivial. However, even for the websites that are unwilling to ask for geolocation information, we think of a method to obtain the best of both worlds. For example, the server may send the original JavaScript code to the users with probability n, and the modified JavaScript code with probability $1 - n$; where n is close to 1. This way the server would have the sample set of distances that indicates whether the IP address is a VPN proxy or not.

- **Proxy adaptation:** We envision once the geolocation check algorithm is known by the VPN proxies, they may start taking some actions to silence the requests for the geolocation URL. For example, they can examine the set of URLs requested by the client, and block the one that sends the information about the geolocation of the client. However, we believe a countermeasure for this type of action is simple enough, that is using rotations. The server can change the URL for this purpose periodically to prevent such an action to achieve its goal. Furthermore, most URLs are complex enough to make it really hard to detect the geolocation information inside any complex URL, making it searching a needle in a haystack.

We are aware of some challenges regarding the distance measured that may cause false positives and true negatives:

- **True negatives:** If the client uses a VPN proxy, and the proxy is really close to the client; VPND may not be able to detect it. Fig. 5 shows the distance between a client from the US and a VPN proxy based in US. It lists the distance between these client and various proxies, indicating there is still a significant distance enabling the detection. The distance without any VPN is just 0.6 miles, therefore the chances of true negatives are pretty low. Moreover, VPND may reduce true negatives by just increasing the threshold for the distance. For example, making it a couple thousand miles would be sufficient that result in blacklisting the proxy only when there is a connection to it from far away countries.
- **False positives:** The traffic from mobile clients may have exit points that are far from the client. Similarly, there may be a significant distance between the client and a NAT device; where VPND may make a mistake and mark a regular traffic as VPN. However, it is easy to detect a mobile client by looking at the User-Agent field in the HTTP request that has the information about the client environment, and remove all false positive threats for mobile clients. On the other hand, we can also reduce the false positive threats for users behind NATs by simply increasing the distance threshold. Fig. 6 shows the distance between a US based client and a proxy server from various countries. The distance in miles basically reflects the geographic distance between the US and other countries. The plot displays three foreign countries such as Netherlands, the UK, and Singapore. As one can see, VPND can reduce false positives increasing the distance threshold that triggers a VPN. For example, a single user from Singapore that share the geolocation information would be sufficient for VPND to detect a US based VPN proxy even if the threshold is thousands of miles.

There are some scalability and performance considerations:

- **Scalability:** VPND completely depends on the external IP information lookup service at ipinfo.io, and we realize if VPND is widely used the load on this service would be overwhelming. However, the service seems to be prepared for high loads and offers various plans to customers. For example, the largest plan costs $\$400$ and it allows for $320,000$ daily requests, which would probably cover a significant amount of traffic.
- **Performance:** We realize depending on the external ipinfo.io may affect the performance of the tool as well. However, we don't necessarily see an extreme necessity for real time analysis. It would be convenient to sample and save packets for an offline analysis, this would allow VPN connections for a short period of time avoiding any performance degradation.

B. HTTP Headers

Using HTTP header fields for VPN detection offers some unique benefits that we did not necessarily anticipate when designing VPND:

- **Extensibility:** Although the HTTP header field mechanism of VPND is currently only built out to support HTTP header field ordering detection for Google Chrome and Mozilla Firefox, it is extensible. This means that by simply adding a list of expected HTTP header field orderings for additional browsers, VPND can be made to support those browsers as well. The unique HTTP header field detection mechanism is also extensible in that the blacklist of know VPN-associated header fields can grow as new fields are detected.
- **No User Interaction:** Because HTTP is a worldwide standard protocol upon which the internet is build, HTTP header field detection should be applicable and possible in most situations. This is in contrast to VPNDs geolocation mechanism, which while arguable more accurate, relies on users to grant permissions.

However, through our experimentation, we have identified certain limitations with relying on HTTP header fields for VPN detection:

- **False Negatives:** When detecting the ordering of HTTP header fields using VPND, we noticed that we were encountering a lot of false negative errors, meaning that some VPNs were not being detected. Upon closer inspection, we realized that the HTTP header field ordering of

these VPNs was very similar to the expected ordering, especially with Mozilla Firefox. In response, we felt that it would be better to have false negatives than false positives because we can layer the detection with other mechanisms. Additionally, if you are going to be adding the IP addresses reported by VPND to a blacklist, it is better to give them the benefit of the doubt.

- **Browser/ User Dependency:** Using HTTP header fields to detect VPN traffic is reliant on the assumption that the expected browser header ordering will remain constant. If something changes with the browser itself, or the user changes their browser settings, there is a potential for false negative reports of VPNs.
- **Lack of Unique Header Fields:** Through our initial packet analysis, we only identified one unique HTTP header field X-Hola-Version which was introduced by Hola VPN proxy. Although VPND does include a mechanism to search for unique, identifying HTTP headers such as this, at this time its effectiveness is minimal.

C. TCP Traffic Pattern

TCP traffic patterns, while fairly consistent in the best case, do have some particular considerations which must be addressed upon usage, mainly concerning consistency, and state maintenance.

- **Performance:** Performance is perhaps this metric's best attribute. The round trip time measurements are fairly easy to manage with low compute power, as it only involves one typical message send, and the tracking of a few time values during the TCP handshake process. However, despite our tool is only looking at one handshake at a time for the sake of clean testing purposes, this extra state must be maintained during the handshake in a real world scenario, and therefore, this may exacerbate the problems that may occur in the case of a TCP SYN flood, or in any similar case where a large volume of requests are incoming, which one server may be unable to handle. However, in normal operating procedures, this is a fairly lightweight detection mechanism, that relies on little outside infrastructure to function.
- **Consistency:** Consistency can be a bit more of a problem with this metric, due to its inherent constraints, and its dependance on VPNs behaving normally, which may not always be the case. First, we made the assumption of using a browser based VPN, which will only route the traffic of browser based activities. This assumption is critical for the usage of a ping, as this allows us to see our difference in round trip time that is necessary for our mechanism to work. Should a VPN route the second part of our ping back through the VPN, or in a more malicious case, induce latency whenever a suspected ping comes through, we may not hit our threshold for double round trip time through VPN usage, and said VPN may go undetected.
  Additionaly, we must stress that VPN traffic that meets this threshold is most consistently apparent through regionally diverse routing scenarios, that cross quite a bit of distance to reach their target. In the case that a VPN route is not very unique, in that the distance to travel is not very far from the actual host origin, and/or the VPN outflow lies near a potential content provider, this may also skew the host throughput in a way that allows certain VPN users to hide from our detection.

However, in a grander scheme, this metric very much suits our Netflix bound targets. To circumvent reigon locking, they would likely have to cross quite a bit of distance to ensure their VPN plants them correctly in an accessible region, thus alleviating our consistency concerns through thoroughly reduced throughput. Also, VPNs for now are very much dumb in regards to our formulas, and would be unable to accommodate for also inducing lag into our probing mechanism, making this point mute for the time being. Finally, good practices to discourage DoS attacks are always encouraged regardless, and the state added from our measurements, a few timestamps which can be dumped post-handshake, is fairly trivial in that regard. We therefore have a fairly solid metric in TCP RTT for addressing our particular target, and can confidently add it to our suite of metric collections for diagnosing VPNs.

VI. Conclusion

VPND is a tool that detects VPN traffic with real time traffic inspection. We identified three opportunities a server side tool could leverage in identifying VPN traffic. Specifically, we made use of geolocation services, HTTP header fields examination, and TCP behaviors as the ways to detect VPN traffic with a high success rate. As discussed throughout the paper, each mechanism itself is trying to increase the success rate on VPN detection. Given we implemented all three mechanisms, we believe collaboratively using all techniques makes it viable to reduce false positives extremely with a maintained high true positive rate. We evaluated our tool with popular VPN proxies, and showed how it takes previous approaches one step further.

References

[1] T. Berger. Analysis of current vpn technologies. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE, 2006.

[2] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*, 2009.

[3] T. Yildirim and P. Radcliffe. A framework for tunneled traffic analysis. In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, volume 2, pages 1029–1034, Feb 2010.