RING-Bearer

Additions to RING: One dVPN to Rule Them All

Julia Odden and Jordan Zax

Prepared for Aleksandar Kuzmanovic and Yunming Xiao
11 June 2021

Abstract

Decentralized VPNs (dVPNs) are a modern alternative to the traditional centralized VPN model. dVPNs use *node machines*, which are managed by *node runners*: users who have chosen to delegate a portion of their own machines' bandwidth to tunneling dVPN traffic. Most dVPNs host their own bandwidth market, where users of the dVPN pay the node runners using various cryptocurrencies. RING is the first and only multi-vendor marketplace for bandwidth serving Mysterium, Tachyon, and Sentinel, three main dVPN providers. RING provides a dashboard that allows node users to easily run their machines as tunnels for all three dVPNs at the same time and to configure policies and network settings for each. RING benefits dVPN providers by widening their pool of available node machines, dVPN users by making the system more scalable and thus more effective, and node runners by keeping the market (and prices) competitive between the three dVPN providers. This report summarizes the shortfalls of both the dVPN infrastructure and RING and proposes additions to the RING dashboard that will help automatically administrate node runners' bandwidth use and maximize their profits. The focus of

our research is on developing policies for dynamic bandwidth allocation within RING to improve the user experiences of node runners.

Introduction

Background on dVPNs

A dVPN is an alternative to the classic *centralized* VPN model. When using a centralized VPN, a user connects to the VPN middlebox, which is administered by the VPN provider and masks the user's IP address, and then indirectly accesses the Internet while being anonymized to their Internet Service Provider (ISP) and to their destination site. Unfortunately, although this model is occasionally helpful in cloaking traffic activities, it is essentially just a redistribution of trust: instead of showing all of their Internet traffic to their ISP, the user is sending it all to the VPN provider via their middlebox. Several major VPNs, including Flash VPN[1] and NordVPN,[2] have suffered major security breaches that revealed stored customer information from their middleboxes. The traditional VPN model works well for circumventing some firewalls and content blocking but does not truly provide anonymity or security to users.

The *decentralized* VPN model, however, makes the vulnerable middlebox obsolete. dVPNs do not have a centralized node or server through which all data passes; instead, they rely on a network of node runners, who allocate portions of their own bandwidth to dVPN providers. The dVPN providers then tunnel the dVPN users' traffic through the node machines using the machines' available bandwidth. If a user wants to access the Internet with a dVPN, they first sign themselves up for one of the three major dVPN providers--Mysterium, Tachyon, or Sentinel--which then connects them to that provider's private fabric of node machines. When the user makes their search or accesses their site, their traffic is divided into subsections, each of which is routed through a different subset of the node network to the destination. With this model, the user's traffic is virtually untraceable: each node machine only tunnels a tiny portion of the user's traffic, all of the traffic is encrypted and unreadable, and there is no central server that sees all of the user's traffic at once or that can store any of the user's browsing data.

Historically, this model has been limited by scalability. There are three distinct dVPN providers, each with its own pool of node machines, and there is currently no incentive for node runners to sign up to run a machine for more than one dVPN provider at once. This is an issue for two reasons: first, the scalability and robustness of each dVPN provider is directly limited by how many node machines they have operating; and second, the node runners should have the flexibility to serve whichever dVPN provider is paying the most at any given time, which would serve the dual purpose of keeping the market competitive and making sure node runners are compensated fairly. The solution to this problem would be a centralized bandwidth marketplace, where each node runner could sign up to provide bandwidth to one, two, or all three dVPN

[1] https://www.welivesecurity.com/2020/07/20/seven-vpn-services-leaked-data-20million-users-report/
[2] https://techcrunch.com/2019/10/21/nordvpn-confirms-it-was-hacked/

providers at once. Ideally, node runners would be able to use the marketplace interface to update traffic settings, price per megabit, content restrictions, and bandwidth limitations for each dVPN. In this scenario, a dVPN user's traffic could be routed through any of the node runners currently signed up to provide bandwidth to the user's chosen provider. A universal bandwidth marketplace like the one described in this work makes the network more scalable for dVPN providers, more fair to the node runners, and more effective for the dVPN users.

The RING Project

RING is the first centralized bandwidth marketplace serving Mysterium, Tachyon, and Sentinel. RING is composed of a front-end dashboard and a back-end infrastructure that enforces the node runners' preferences. The RING dashboard allows node runners to provide bandwidth to all three dVPN providers at once, set their bandwidth limits interactively throughout their sessions, place restrictions on the types of traffic that are allowed to move through their system, and configure their price settings. Any traffic routed to a node machine that violates any of the user's preferences (e.g. a type of traffic that the user has banned, or traffic that uses more than the node runner's allotted bandwidth) is simply dropped at the node machine without being served, thus honoring the runner's restrictions. RING encourages the universal bandwidth pool model by making it trivial for node runners to share bandwidth with all three dVPN providers at the same time, meaning that most RING users will be a part of the node fabric for all three systems.

Shortfalls of RING

RING is currently in its beta phase and still has several bugs that need to be resolved, namely in security, robustness, and reliability. The authors will focus on the issue of dynamic bandwidth allocation. At the moment, when a node runner activates their machine with RING, they can set custom preferences for bandwidth limits (e.g., limiting all of the dVPN providers to 5 MBPS on their machine). These limits are static, meaning that they will persist at a single value (e.g., 5 MBPS) for the duration of the session, unless the node runner manually changes the bandwidth limit. This is highly inconvenient: for example, if a node runner allocated 10 MBPS to Sentinel one morning because they were not using any bandwidth themselves, but later decided to stream a movie, the bandwidth use by the dVPN might increase their network traffic enough to disrupt their viewing experience if they did not remember to change the limit to something lower or deactivate their node machine. RING has no built-in system for dynamically allocating bandwidth based on any input, be it a set schedule, network-traffic monitoring, or even price monitoring between the dVPN providers.

Ideally, in its final form, RING will become more than a universal bandwidth marketplace: it will serve as an automatic bandwidth broker, maximizing node runners' profits while minimizing conflicts on their networks. To get to this point, RING needs four primary improvements:

1. **A preliminary schedule-based dynamic bandwidth allocation algorithm** that provides a proof of concept for the dynamic allocation model.

2. **A network-traffic-aware dynamic allocation algorithm** that automatically assigns bandwidth caps for each provider based on the current traffic on the node machine's network to minimize congestion.
3. **A profit-maximizing dynamic allocation algorithm** that assigns bandwidth between the three dVPN providers under a predetermined limit (in MBPS) based on which provider's cryptocurrency is worth the most.
4. **A synthesized, hands-free dynamic allocation algorithm** that calculates bandwidth limits based on step 2 and allocates bandwidth within that limit between the three dVPN providers based on step 3.

These four improvements to the RING implementation will make node running a hands-free experience, in which RING both prevents instances of network congestion and maximizes the node runners' profits without any human intervention.

RING-Bearer

The authors propose three additions to the RING system, following the outline described above: a schedule-based bandwidth allocation system, an automatic network-traffic-aware bandwidth allocation system, and a price-based bandwidth distribution system. For this work, the authors implemented and tested the first of these three algorithms. In this section, we will describe our implementation and test results, as well as lay out our recommendations for future work. At the time of writing, the primary RING developer, Yunming Xiao, has already begun work on a price-aware distribution system, and we respect that our outline might not agree fully with his implementation and should be taken as a recommendation as opposed to strict guidance.

Schedule-Based Bandwidth Allocation Policy

The schedule-based bandwidth allocation policy is built on the idea that a node runner is generally aware of their bandwidth needs at different times of day and can set limits on how much bandwidth the dVPNs can take up on their machines accordingly. For example, if a node runner knows that he works from 9-5 but rarely uses any bandwidth outside of those hours, he might be able to allocate 10 MBPS of bandwidth to the dVPNs from 5pm to 9am and then zero MBPS from 9am to 5pm. By increasing the dVPN bandwidth utilization when the node runner is not active on the network and decreasing it when they are, RING will help the node runner maximize their profit while simultaneously minimizing network congestion caused by the dVPN traffic.

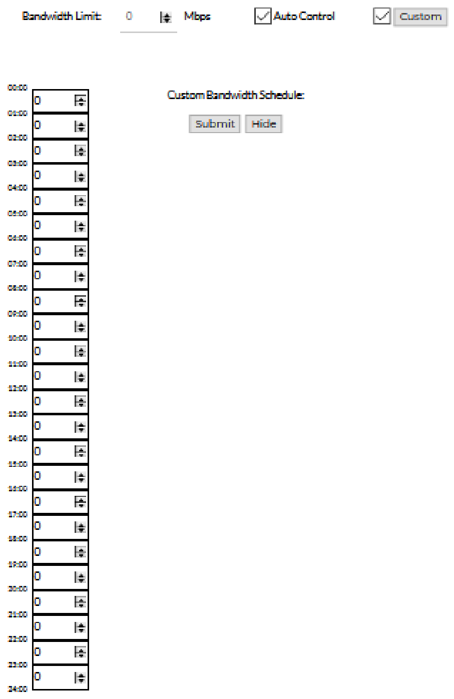
Implementation

The authors provide an implementation of this schedule-based bandwidth allocation policy. Our solution is compatible with the existing RING framework and will still permit node runners to customize their price settings and content restrictions. RING is currently only compatible with Ubuntu, so we have written our additions to fit within this framework.

To implement our bandwidth scheduler, we first had to change the front-end web page in order to allow a user to input a schedule. This was quickly done using HTML, CSS and JavaScript. The previous implementation of the custom price policy was followed closely in order to implement this part. From this webpage, information containing bandwidth policy selections are sent via an HTTP POST request to the backend.

The backend API was changed in order to accept this new POST request to create a custom bandwidth schedule. From here, we were able to send this updated information to the controller in order to change the bandwidth limits on whichever indicated dVPN provider. Some code in the controller runs every 30 minutes or so, thus here, we check to see what the bandwidth limit should be and use some previously written code to change the bandwidth limit according to the schedule sent over by a node runner.

User Interface


Bandwidth Limit: 0 Mbps ✓ Auto Control ✓ Custom

Custom Bandwidth Schedule:

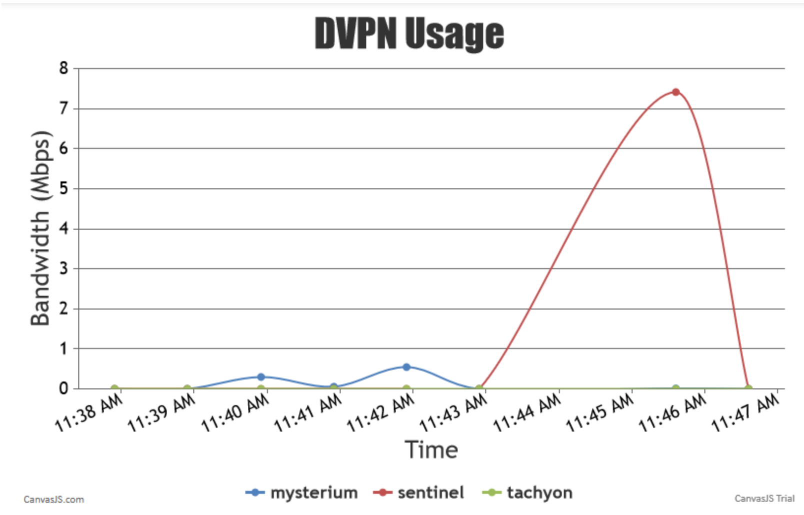
00:00	0
01:00	0
02:00	0
03:00	0
04:00	0
05:00	0
06:00	0
07:00	0
08:00	0
09:00	0
10:00	0
11:00	0
12:00	0
13:00	0
14:00	0
15:00	0
16:00	0
17:00	0
18:00	0
19:00	0
20:00	0
21:00	0
22:00	0
23:00	0
24:00	

The figure to the right shows the RING-Bearer user interface, implemented in the RingBearer branch of the RING GitHub repository (linked here: https://github.com/yunmingxiao /RING/tree/RingBearer). The node runner has the option to set a bandwidth limit for every hour, starting on the hour; for example, if they were to put a 1 in the topmost box, RING would respect a bandwidth limit of 1 MBPS from midnight until 1AM. The interface accepts a high level of granularity, allowing users to input limits down to fractions of MBPS. It will not accept non-numeric input, leaving it less vulnerable to buffer overflows and other malicious inputs. Once set, the schedule will persist in the node runner's RING settings until they either manually reconfigure the schedule or shut down the RING instance. During this time, the node runner will not have to monitor RING's bandwidth consumption. This policy can be set individually for each of the three available dVPN providers, giving node runners the flexibility to allocate more bandwidth to dVPNs whose share prices are higher at any given time.

Testing

At the time of writing, we have performed preliminary testing on our implementation of the schedule-based policy. We ran our code on two Amazon EC2 instances, each with 20+ GB of onboard hard disk memory, and continually monitored the bandwidth use by each dVPN via the RING Graphical User Interface (GUI) to ensure that our limits are being respected. We expect to see RING consistently respecting the test-input bandwidth caps, and to see the amount of bandwidth being used by each dVPN throughout the day changing in accordance with our test policy.

Testing our solution in this way is difficult and prone to error for several reasons. First, the GUI does not provide extremely fine-grained feedback; as demonstrated in the figure below, the GUI only updates once every few minutes. We assume that the graph generated by the GUI is a good representation of average traffic over a long period of time (i.e., >= 1 hour), but we do not know for sure that our system is not experiencing bandwidth spikes in between measurements. For a more detailed result, we would need to log precise measurements at a smaller interval (i.e., every second) and graph those as well; this solution is infeasible at the moment for reasons outlined in the *Overhead and Shutdown* section of this document. Second, the actual demand on the node machine might at any given time be lower than the node runner's bandwidth threshold. In our research, we rarely see sustained spikes in bandwidth use by the dVPN of over 5 MBPS. If a node runner were to set their bandwidth cap at 10 MBPS, the RING GUI might show that bandwidth cap being respected, when in reality there is simply never 10 MBPS of demand on the node machine.



We set up our testing environment to try to minimize these two sources for error. To minimize the effects of low-granularity readings, we ran prolonged overnight tests that would better demonstrate the trends in the bandwidth usage. To ensure that our good results were not a product of low demand, we consistently set our policy to be below our observed average traffic for our chosen provider (~5 MBPS).

We also propose several future phases of testing for our implementation that were out of the scope of this work for financial and time-related reasons. First, we advise activating several more Amazon EC2 instances and running RING-Bearer on each of them with very low and very specific bandwidth constraints (<= 1.5 MBPS). This would provide a better indication of the actual performance of our policy in practice. Following this stress test, we recommend deploying our policy to actual node runners and performing both a qualitative and a quantitative analysis of the node runners' user experiences. Some survey questions we would ask are as follows:

1. Do you observe the policy you set being respected?
2. Do you notice less traffic on your network when RING-Bearer is handling your dVPN bandwidth allocation? (RING developers could potentially see better answers to this question if they also provided a network speed-testing framework.)
3. Do you find that you spend less time manually configuring your bandwidth limits than you did before? If so, by how much?

We hypothesize based on preliminary testing that the answer to question 1 will be a resounding *yes*, since it is simply a matter of making sure that our code works bug-free. For question 2, we anticipate an observable reduction in network traffic over simply maintaining a high, static bandwidth cap for dVPN traffic. For question 3, we anticipate that RING-Bearer will significantly reduce the amount of time users spend manually configuring their policies.

Results

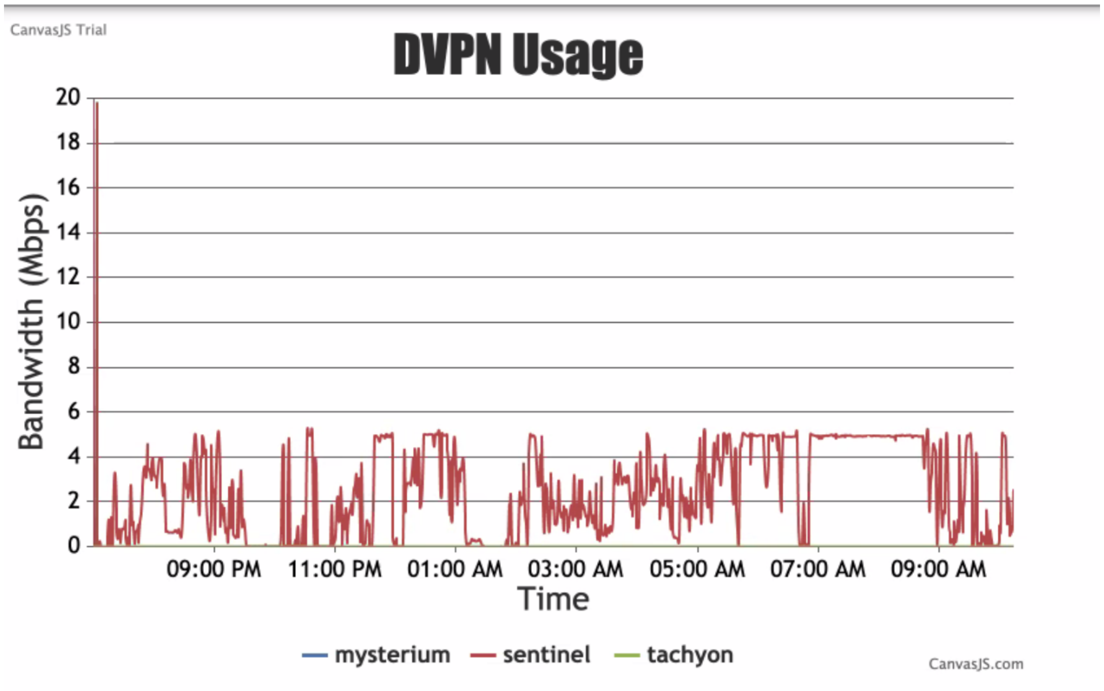
We ran two distinct tests with two separate policies to demonstrate our work.

Policy 1

Row 1: Mysterium, Row 2: Sentinel

8 PM	9 PM	10 PM	11 PM	12 AM	1 AM	2 AM	3 AM	4 AM	5 AM	6 AM	7 AM	8 AM	9 AM
0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	5	5	5	5	5	5	5	5	5

Our first test was an extremely simple context-based experiment. We left Sentinel with a persistent 5 MBPS bandwidth cap using the existing static bandwidth limit framework for context, to demonstrate that dVPN bandwidth was indeed being used. We then used our policy to set a 0 MBPS bandwidth limit on Mysterium, hour by hour. We did not activate an instance of Tachyon. The figure below demonstrates the results of a twelve-hour test.



This graph, though demonstrating a simple policy, shows that RING-Bearer is effective at enforcing the user-defined bandwidth limits. There is no traffic on Mysterium, even though Sentinel's bandwidth usage indicates that there is dVPN traffic being handled.

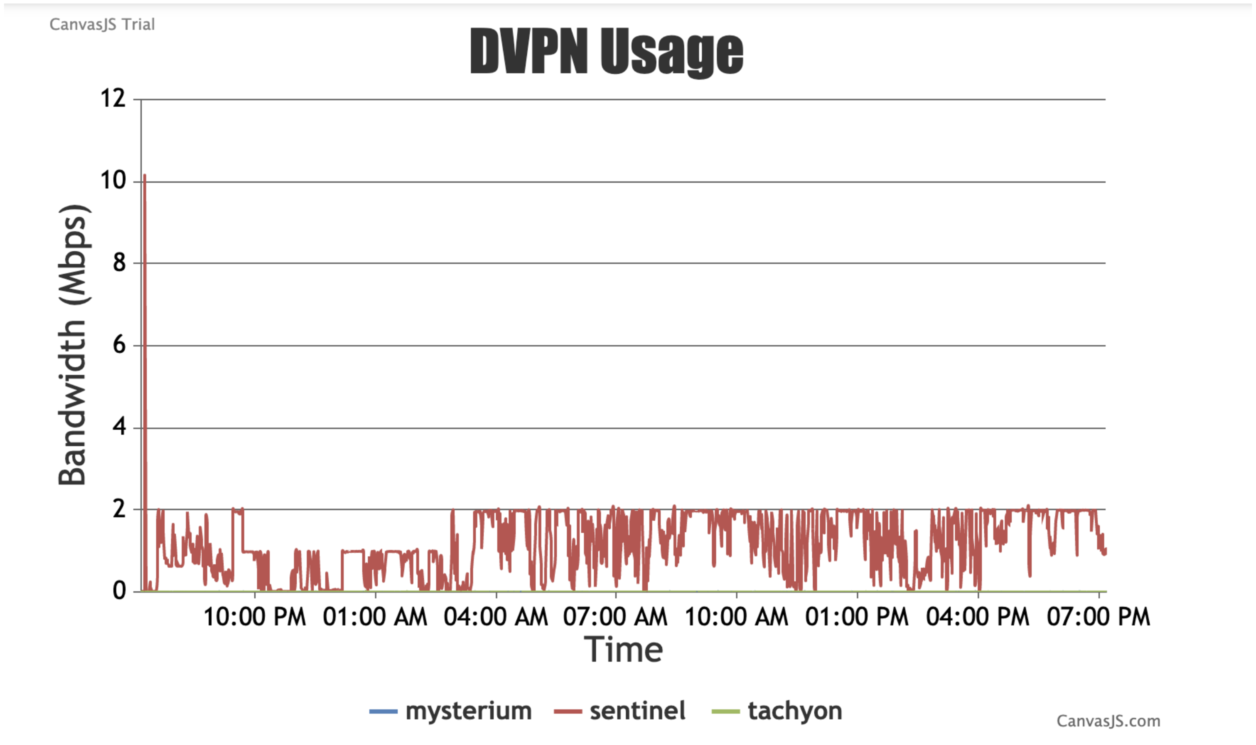
Policy 2

Row: Sentinel

00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00	10:00	11:00
0	0	1	1	1	0	0	2	2	2	0	0

12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00
0	0	0	0	0	0	0	2	2	2	0	0

The policy above is slightly more complicated than the last one. We set a hard limit of 0 MBPS on Mysterium and Tachyon and the dynamic policy above on Sentinel. As you can see in the figure below, we had some interesting results: it appears that RING-Bearer honored every limit that was *not* zero, but did not appropriately cap Sentinel's bandwidth at zero when asked. Instead, it persisted with the previous non-zero bandwidth cap until the next non-zero bandwidth cap was reached, at which point it switched to the next one (see 02:00 and 07:00 on the graph below). Note that the times don't line up; this is because RING is running in CST, while our EC2 instance is running in a different time zone, five hours ahead.



We wonder if this is a result of RING ignoring bandwidth caps set to zero due to some sort of policy default (i.e., if the bandwidth is set to zero, the code does not make any changes to the policy).

Network-Traffic-Aware Bandwidth Allocation Policy

Our second proposition after the schedule-based policy is the network-traffic-aware policy. The implementation of this step was beyond the scope of our work, but we believe it is the next logical step in the development of RING-Bearer.

Our proposed system would automatically adjust the overall bandwidth cap on all three dVPN providers put together based on the current efficiency of the node runner's system, accounting for general connection quality as well as network congestion. It would then deterministically and equally distribute bandwidth within that upper bound to each of the three dVPN providers. If the node runner were trying to stream a movie, this policy would be able to detect their high bandwidth demand and scale down the amount of bandwidth available to the dVPN providers accordingly to improve the node runner's viewing experience. Similarly, if the network is relatively empty, the policy should provide much more bandwidth to the dVPNs to both support the providers and increase the node runner's profits.

We expect this could be achieved using an architecture similar to the one used by https://www.speedtest.net/, which provides live analysis of ping, download, and upload speeds on a user's device. These measurements directly indicate the network efficiency and indirectly provide feedback on the network traffic. Each of these measurements, especially ping speed, can be taken by different Python libraries and command line scripts.

This policy will be the first step toward a truly hands-free experience of node running. Additionally, after this policy is implemented, RING will be in an ideal state to be preliminarily deployed for commercial use; our next policy, a price-based bandwidth distribution algorithm, is not currently relevant to the space, for reasons discussed below.

Price-Based Bandwidth Distribution

As mentioned previously, Yunming Xiao has already begun work on a price-based bandwidth distribution policy. We specify that this is a *distribution* policy as opposed to an *allocation* policy because it is intended to distribute bandwidth below a certain pre-determined cap among the three dVPN providers, as opposed to allocating bandwidth on a node machine to RING as a whole. This policy is a thing of the future because its main goal is to distribute the most bandwidth to the dVPN provider whose chosen payment cryptocurrency is worth the most; however, at the moment, only Mysterium is actually compensating node runners for their bandwidth.

The three providers all use different cryptocurrencies to compensate node runners. We want RING-Bearer to be able to observe the current market prices of each of the cryptocurrencies and automatically distribute bandwidth to the provider with the highest-valued crypto. For example, if Mysterium's cryptocurrency payment was worth $15 per share, but Tachyon's and Sentinel's were worth $3, and the user had set a bandwidth cap of 5 MBPS overall, RING-Bearer should give all 5 MBPS of bandwidth to Mysterium and set the bandwidth limits on Sentinel and Tachyon to zero. This policy has two components: an optimization algorithm and a cryptocurrency stock monitor. This policy would also automatically set the node runner's bandwidth price per megabit; this number is up to the node runner's discretion and defines how much they are paid per megabit, but also correlates to how much bandwidth is passed through their machine.

Synthesis

In the end state of this project, we would like to synthesize the network-traffic-aware allocation policy and the price-optimizing distribution policy into one master policy that would provide a completely hands-free node running experience. The automatic allocation policy would handle calculating and enforcing the overall bandwidth cap on all three dVPN providers combined, while the distribution policy would divide the available bandwidth among the three providers to maximize the node runner's profits. In this system, a node runner would be fully incentivized to activate their machine on all three dVPN instances at once, thus making each service more scalable and keeping their pricing competitive.

Overhead and Shutdown

In the process of our research, we encountered a problematic bug that we wanted to document. The RING interface is configured to make periodic writes to log files; as mentioned previously, we would have liked to log some more accurate test results of our own using this system. We attempted to run a twelve-hour test on an EC2 instance and encountered several issues with this system, namely that the log outputs are so verbose that they rapidly drain the system memory. At the end of our test, we attempted to shut down the RING instance using ctrl+C, as is standard in a shutdown. When we rebooted the EC2 instance approximately 48 hours later, RING had used the entire hard disk memory of the machine in log files. That is at least *20 gigabytes* of memory in two days, probably more; we are unsure how much memory the provided EC2 instances had to begin with. Because of the overabundance of output to the log files, we declined to log more detailed MBPS readings to avoid burning through the memory of the test machines and necessitating another reset.

Upon examining the *ps aux* output on the instance, we noticed that there were several instances of the *bash ./start.sh* command active even after our attempted shutdown, as well as several *sleep* commands and one instance of the *python webdriver.py*, also a RING process. In order to properly kill RING, the node runner has to manually use the *kill* and *killall* commands on every main process and child process. This is the reason that the program continues logging after kill. We strongly recommend updating the kill script to universally end the process and stop the logging. We also recommend reducing the size and frequency of log events to lighten the memory overhead of RING. Once this is complete, we recommend logging more detailed bandwidth use readings and re-running the authors' tests on the schedule-based bandwidth allocation system to confirm its efficacy.

Conclusion

We propose RING-Bearer, a set of additions to the RING framework designed to provide node runners with a completely hands-free experience of dVPN tunnelling. In our work, we provide an implementation and preliminary testing of a schedule-based bandwidth allocation system and a detailed proposal for future policies that aim to reduce network congestion on the node machines while maximizing the node runners' profits. We also critique the memory overhead of RING and offer suggestions for improvement. In the future, the developers should take steps to perform wide-scale testing of our schedule-based allocation system, integrate our proposed network-traffic-aware bandwidth allocation component, complete their price optimization component, and then integrate those last two components to make RING fully self-sufficient. With RING, dVPNs will be vastly more scalable, competitive, and speedy, allowing dVPN providers to meet demand while simultaneously treating node runners fairly and providing quality service to users. RING-Bearer is the component of the project that makes the system less invasive and more profitable for node runners.

Acknowledgements

We would like to thank Yunming Xiao for his guidance with this project.