Improving Onion Routing: A Cloud-Based Approach

Jack Good
Computer Science
University of Virginia
Charlottesville, Virginia US
jg8dp@virginia.edu

Brandie Young
Computer Science
University of Virginia
Charlottesville, Virginia US
bay5fb@virginia.com

Vineet Kalpathi
Computer Science
University of Virginia
Charlottesville, Virginia US
vsk6ab@virginia.edu

ABSTRACT

The Onion Router (TOR), a free worldwide network run by volunteers, allows its users to access the internet anonymously by obfuscating their connection. The Onion Router enables the proliferation of cybercrime such as illegal markets and pedophilia rings; however, in an age of increased internet censorship, data mining, and surveillance, these systems also grant users with increased security, privacy, and freedom especially in cases of state oppression. The Onion Router is not without its drawbacks; TOR's network can exhibit poor performance, reliability, and has the potential to be blocked. The advent of cloud computing, which has the ability to solve TOR's shortcomings, allows for on-demand provisioning of massive computing performance and connectivity. A cloud-based Onion Router (COR) will offer better performance, connectivity, and deter censorship. How can the security of the TOR network be strengthened by the use of cloud technologies, for the purpose of protecting the privacy of users on the Internet and promoting Internet freedom? Current research on this topic shows that a cloud-based onion router network is feasible, but has left the tradeoffs of cloud computing configurations for future research. The purpose of this technical report is to examine the tradeoffs of network topology, instance purchasing options, instance family types, typical applications on TOR and their required bandwidth, and regions and availability zones. We aim to prescribe the most optimal configuration based on latency, throughput, monetary cost per user, usability, and of course the preservation of security from local and global network adversaries.

1 Background

Although the dark web's volunteer-based infrastructure model preserves anonymity through its decentralized nature, it also poses adverse effects on the network's overall performance and security. There are a limited number of volunteer-run relays located worldwide, subject to highly variable network performance depending on the host's location and Internet Service Provider plan. Relays with limited access to network bandwidth create bottlenecks within onion-routed circuits, negatively affecting the latency of TOR connections [16]. Additionally, the dark web relies on a few well-known entry nodes for onboarding users to the network, which allows for network administrators to easily censor content or block all anonymous traffic by blacklisting all known TOR addresses [17]. This is also how authoritarian governments can censor the spread of information amongst and beyond its populations, stifle criticism, and ultimately oppress citizens.

Several studies suggest the potential for cloud infrastructure to greatly mitigate these performance and security issues caused by TOR's current infrastructure model, presenting an incredibly efficient and secure method of browsing the internet freely. Thus, the objective of our project is to explore the fairly novel design space of applying cloud computing to TOR in order to improve performance, connectivity, and deter censorship. Our technical research will utilize existing literature, documentation, and data to analyze the tradeoffs of different types of cloud compute resources and network configurations and their effect on the key metrics of secure communications within a cloud-based onion routing (COR) network.

2 Related Work

The sheer scalability and elasticity of services presented by major cloud hosting providers (CHP), such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud, could tremendously enhance TOR's user experience. An experimental implementation and analysis of COR yielded results that exhibit client download times 7.6× faster than TOR [16]. Another small-scale implementation that utilized dynamically-addressed virtual machine (VM) relays proved to be highly tolerant against denial-of-service attacks, given that CHPs can simply spin up new VM relay instances to handle overwhelming amounts of traffic [18]. These initial results are extraordinarily promising; however, these experiments fail to encompass the added complexity of involving CHPs in the anonymity network's infrastructure at a large scale.

The preservation of anonymity provided by TOR lies within the trust of users and the volunteers running and maintaining the network's relays. A cloud-based model adds several relationships to the picture, including the relationship between CHPs and end users. Given that direct payment systems between CHPs and users could completely deanonymize all network activity and render all onion routing useless, a large concern for implementing COR is deciding who is responsible for the cost of running relays provided by CHPs. Jones et al [16] suggest the need for an additional entity, called an "anonymity service provider" (ASP), to purchase relays and provide a secure, anonymous transaction for users to pay for access to these nodes. Additionally, a fully anonymized implementation of COR requires the existence of multiple ASPs to ensure that no single entity has the ability to oversee and discern which relays are carrying a given user's traffic.

3 System Design

A robust COR implementation will utilize computing resources from multiple CHPs, however for the purpose of this research paper we will focus on offerings from the CHP with the largest market share, AWS. Our criteria for choosing instance options include monetary cost per user, sufficient compute, disk storage, RAM, and bandwidth. When analyzing networking aspects of COR we consider sender anonymity, receiver anonymity, performance, and security. We now investigate instance purchasing options, instance family types, typical TOR applications and required bandwidth, regions and availability zones, and network topology.

Table of Contents

3.1 Instance Purchasing Options

There are various instance purchasing options to consider for the use case of COR. To start off, we look at on-demand instances. At first glance, these make the most sense for our COR use case. They can be quickly spun up or blown away, and the ASP will only pay per second of use, effectively scaling the relay fleet with the amount of COR network traffic [1]. Upon being spun up, an instance will also have a new IP address, which is useful in that it will by default prevent wholesale blocking from censors attempting to blacklist known TOR addresses. Prices of on-demand instances are volatile, subject to the fluctuating supply and demand of the instance market. Historically, the price of instances has dropped with the booming success of cloud technology and providers cutting prices, competing for customers [12].

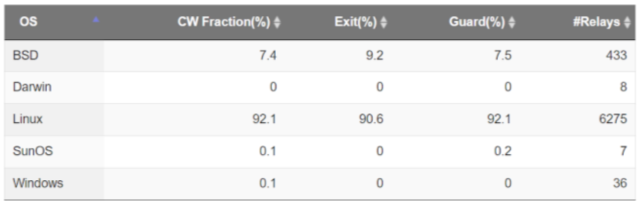
Next, we look at reserved instances. Reserved options allow the purchaser to "lock in" a cheaper price than the on-demand price, by committing to an instance type and Region for a period of 1-3 years. With reserved instances, there is full upfront, partial upfront, and no upfront options. Generally speaking, you can save more money making a higher upfront payment for a reserved instance. With Reserved Instances, you can save up to 75% over equivalent on-demand capacity. The reserved option is tempting, as it locks in a much lower cost for users of the COR network, and would likely make the COR network more appealing to more users, thereby further driving down the price. However, upfront RI options would require ASPs to somehow fundraise enough money to be able to pay for the full year of the reservation, meaning enough tokens would need to have been purchased in time. That brings up the issue—what if not enough money is on hand for the ASP to pay? And even if a less upfront option was used, the issue still exists; what if there is not enough money on hand to pay the hourly rate of the no upfront RI? This option is ultimately somewhat of a problem because the number of reservations made at the yearly turnover is not dependent on the real-time, potentially constantly changing demand of the COR network. It's also important to note that the IP address of the reserved instances would only change when it was stopped or hibernated (which is important to deter censors attempting to blacklist known TOR addresses) so there would need to be some extra overhead to ensure the stopping/restarting of the instance [1].

Lastly, we look at spot instances. Similarly to on-demand instances, you can cancel a spot instance at any time and you only pay for per second of usage. However, spot instances offer the largest potential discount from on-demand prices, up to 90% in the right conditions. Spot instances prices are extremely volatile (updated every five minutes), and represent the excess capacity Amazon has on hand, in case of surges in customer demand [10]. So the low price has a cost—it's that Amazon can "pull the plug" and terminate spot instances with just a 2 minute warning. This may be doable with the COR network, but it would mean that this instance would need to be quickly removed from the pool of available relays, so that traffic is not lost. It would also require extra overhead in the form of a protocol to notify clients that are attempting to route through this instance, that it needs to reroute its traffic through other relays. Another worthwhile thing to consider is that there wouldn't always be Spot instances available for the ASP to bid on, so some kind of backup plan would be needed when there aren't Spot instances available—perhaps backfilling with on-demand instances. Ultimately, Spot instances seem extremely attractive as a way to make the COR network as cheap as possible for users.

3.2 Instance Family Types

Within Amazon EC2, there exists a number of instance families an ASP can consider for use as a COR relay. Each type of instance offers a distinct set of computational resources to cater to varying general purpose, compute-optimized, memory-optimized, and storage-optimized use-cases [2]. An analysis of COR relay resource usage is dependent on the function of the node within the onion routing algorithm. TOR's routing scheme currently uses three primary types of relays within a single circuit: entry (or guard) relays, middle relays, and exit relays. In a typical connection, clients connect directly to a guard node, which routes traffic through one or more middle relays, subsequently accessing a destination through an exit node. This separation of functionality creates varying resource considerations for each type of relay, given that each relay faces slightly different requirements regarding the handling of user traffic. Therefore, our investigation of potential instance types for use as a COR relay must evaluate operating system, compute, storage, and memory requirements for each type of node. We utilize the relay requirements expressed by the TOR Relay Guide on torproject.org [23].

Like most modern software, onion routing is completely independent of the operating system it runs on. The TOR community recommends that relay administrators run the operating system that they are most familiar with, in an attempt to further diversify and decentralize the set of nodes that run TOR, thereby improving the overall security of the network. Figure 1 lists the current distribution of operating systems across relays on the TOR network.



OS	CW Fraction(%)	Exit(%)	Guard(%)	#Relays
BSD	7.4	9.2	7.5	433
Darwin	0	0	0	8
Linux	92.1	90.6	92.1	6275
SunOS	0.1	0	0.2	7
Windows	0.1	0	0	36

Figure 1: Operating System Distribution Across the Onion Router Network [19]

It can be observed that the large majority of TOR relays run Linux (specifically Debian) images. In the context of COR, this recommendation gives ASPs free reign over the operating systems run on the relays they provide, allowing for a much more even OS distribution across COR nodes. Amazon EC2 offers seven different Linux images, four Microsoft Server images, and an option for custom images for users to choose from. Microsoft and custom images cost slightly more than Linux alternatives, illustrating a tradeoff between cost and the added security from a diversification of relays. An ASP must therefore decide on an optimal ratio of relay OSs to maximize security while minimizing cost for themselves and the end user. However, the ease of deploying COR nodes with a large assortment of operating systems exhibits a security advantage over TOR's current infrastructure.

Relays on the TOR network do not have stringent CPU requirements, given that most modern processors can easily handle running TOR on a single server. The only recommendation for relay processors is support for Intel's Advanced Encryption Standard New Instructions (AES-NI), an instruction set that accelerates data encryption on contemporary intel processors [15]. This feature improves overall relay performance, allowing for approximately 400-450 Mbps upstream and downstream connections [23]—greatly exceeding the minimum requirement of 10 Mbps, the recommended non-exit relay bandwidth of 16 Mbps, and the 'fast exit relay' requirement of 100 Mbps. Fortunately, all EC2 instances support AES-NI and therefore meet the compute requirements of COR guard, middle, and exit nodes [2]. Similarly, relays do not typically utilize a large amount of disk storage due to the dynamic nature of onion routing. The algorithm itself does not require more than 200 MB of disk space to operate, allowing for all instance families to meet storage requirements for COR operation.

Memory requirements for COR relays are minimal, relative to the options that CHPs provide for compute instances. Guard and middle nodes require at least 512 MB of RAM for relays that support less than 40 Mbps connections, and 1 GB for relays that support 40 Mbps or higher. Given that all EC2 instances support a minimum of 400 Mbps, we can assume a 1 GB RAM requirement for non-exit COR relays. Exit nodes require at least 1.5 GB of RAM per instance [23]. This differentiation stems from the separation of relay functions; non-exit nodes are only responsible for routing and obfuscating traffic via encryption, while exit nodes have the added responsibility of maintaining the users' intended requests and responses. Furthermore, all non-nano EC2 instances meet the memory requirements for COR relays, offering a large selection of low-cost options for ASPs to provide relay access to users.

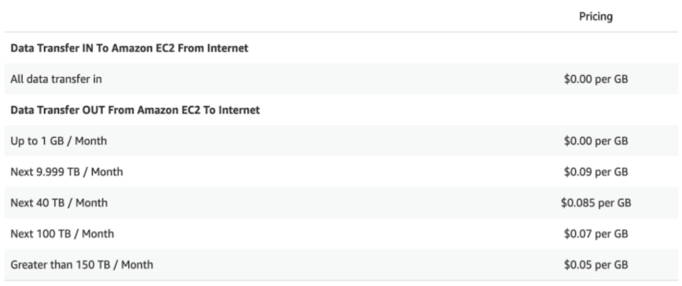
Given the resource considerations for guard, middle, exit, and bridge relays, ASPs have a considerable amount of flexibility in choosing which instance types to use for COR nodes. Memory is the only limiting factor in choosing an instance; therefore, assuming an on-demand pricing model, ASPs could minimize costs by utilizing the T-family of instances for use as COR relays—the size of the instance remains contingent upon network bandwidth requirements, which will be discussed in the next section. However, the anonymity of COR relies on the existence of multiple ASP entities, giving rise to inherent competition among such institutions. This competitive force presents an additional motivation for ASPs to provide greater performance and security to COR users by offering compute-optimized relays or a larger variety of operating systems. Although our analysis exhibits the lowest-cost options for ASPs and end-users, there is an immanent tradeoff between cost and performance when selecting instance types for COR relays. Anonymity-providing entities must consider providing relay access with higher performance and security to appeal to the myriad of COR users.

3.3 Typical TOR Applications and Bandwidth

At this point, we have found that T-family spot instances will be useful in a COR implementation, but there is one more benchmark to consider when prescribing the most practical instance type. When appropriating the egress/ingress capabilities that a TOR relay should have, we investigate the average bandwidth usage of a TOR relay and resulting cost of this in the cloud, mostly commonly used services used in the Dark Web, and finally, which instance types would best fit the COR use case in terms of bandwidth.

To start off, note that Amazon charges nothing for downstream traffic and traffic between its own data centers. However, we must take into account upstream traffic and traffic between relays in different Cloud Hosting Providers. This yields another piece of consideration when trying to determine the number of CHPs to route through—it adds another upstream connection, thereby increasing bandwidth costs. For example, in a COR circuit with 2 nodes in different CHPs, we have 2 upstream connections and 2 downstream connections in our route to appropriate bandwidth to [16].

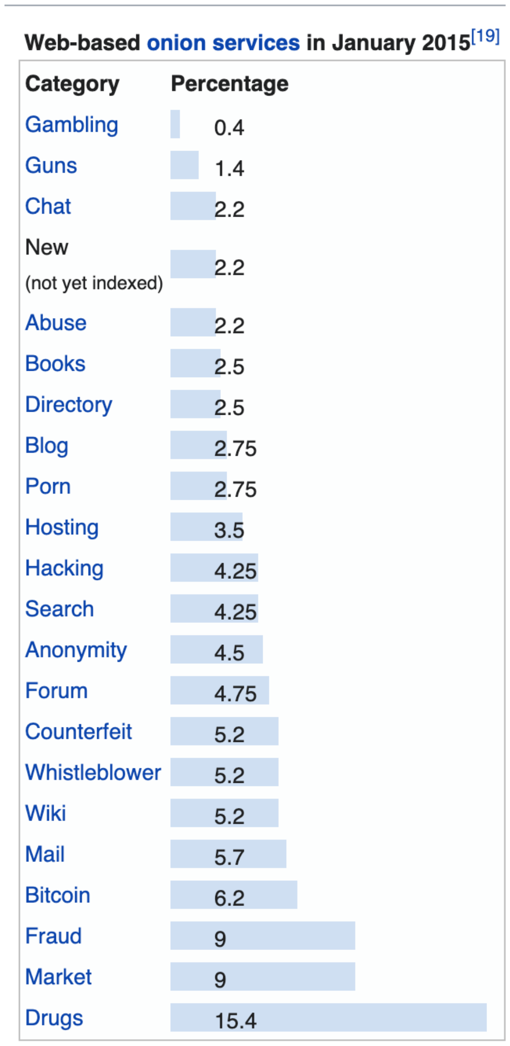
By looking at the metrics provided by torproject.org [26], the upper bound of traffic through a TOR relay is .9 Gbits/s which equals 112.5 MB/s in November of 2020. To determine upstream traffic, we take roughly half of this, so on average 56.25 MB/s. For a month of usage, we get 56.25 MB/s * 3600 s/hr * 24 hr/day * 31 days/month = 150,660,000 MB = 150,660 GB, which is approximately 150 TB/month of traffic.



	Pricing
Data Transfer IN To Amazon EC2 From Internet	
All data transfer in	$0.00 per GB
Data Transfer OUT From Amazon EC2 To Internet	
Up to 1 GB / Month	$0.00 per GB
Next 9.999 TB / Month	$0.09 per GB
Next 40 TB / Month	$0.085 per GB
Next 100 TB / Month	$0.07 per GB
Greater than 150 TB / Month	$0.05 per GB

Figure 2: Cost of Data In/Out of Amazon [3]

To estimate what this would cost with COR we use Amazon's pricing. Amazon uses a pay-as-you-go, sliding-scale for upstream traffic pictured above in Figure 2. Therefore, we have $1500 for the first 10 TB, $4400 for the next 40 TB, and $9000 for the last 100 TB. That brings us to $14,900 to run an Amazon EC2 instance at the current bandwidth requirements. Note that these prices, like that of the actual instances, have been on the decline over time, making the cost of running these relays even cheaper over time.



Web-based onion services in January 2015[19]

Category	Percentage
Gambling	0.4
Guns	1.4
Chat	2.2
New (not yet indexed)	2.2
Abuse	2.2
Books	2.5
Directory	2.5
Blog	2.75
Porn	2.75
Hosting	3.5
Hacking	4.25
Search	4.25
Anonymity	4.5
Forum	4.75
Counterfeit	5.2
Whistleblower	5.2
Wiki	5.2
Mail	5.7
Bitcoin	6.2
Fraud	9
Market	9
Drugs	15.4

Figure 3: Percentages of Web-Based Onion Services

Figure 3 shows the percentages of various web-based onion services, in which Dr. Gareth Owen [20] in one of the largest studies into TOR to date, categorized known sites based on content. This indicates that the most TOR services are sites that are associated with drugs, the black market, fraudulent papers, bitcoin, and mail.

We dive deeper. Given each of these categories, Figure 4 [20] shows us what proportion of directory requests went to each of these categories. These percentages of "successful hidden service requests" likely loosely correlate with visits by users (although we can't know for sure the proportion of users vs. automated requests by nature of the dark web). These numbers may also be inflated by "crawlers", groups that visit these sites on a regular basis, for example child protection agencies. Figure 2 would indicate an alarming percentage of requests went to web-based services about abuse, one would hope the numbers are inflated by these child protection agencies.

Following abuse, the other top requested services look to be porn, search, forum, wiki, market, and drugs.

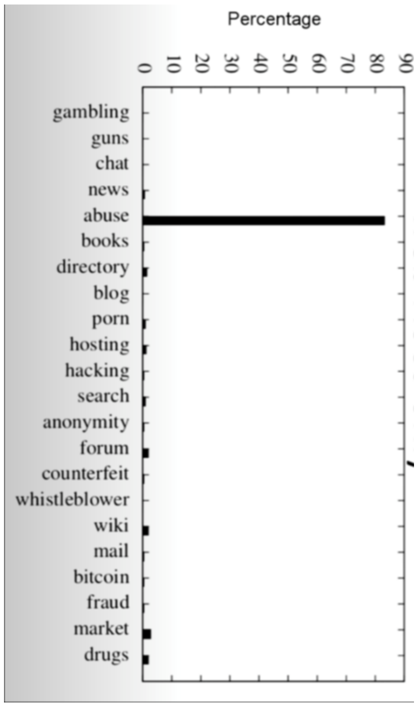
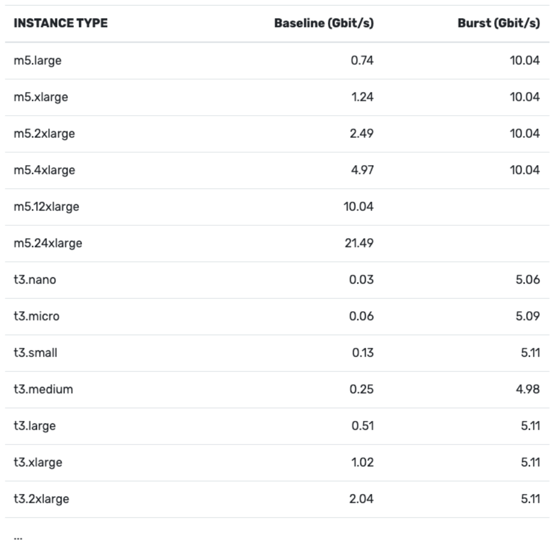


Figure 4: Content vs. Popularity in Web-Based Services [20]

These services would all have various bandwidth requirements per user, most no different than a normal website, but some on par with that of a Netflix or other video streaming service.

In light of this information and how much bandwidth a common TOR relay utilizes, we investigate which instance size would fit our bandwidth. Note that AWS does not disclose the network capacity of all their instance types in detail. But we can find baseline throughputs for various instance types as found in practice, see Figure 5 [27]. Note that as a general rule, larger instance types have increasingly larger baseline Gbit/s network throughputs.



INSTANCE TYPE	Baseline (Gbit/s)	Burst (Gbit/s)
m5.large	0.74	10.04
m5.xlarge	1.24	10.04
m5.2xlarge	2.49	10.04
m5.4xlarge	4.97	10.04
m5.12xlarge	10.04	
m5.24xlarge	21.49	
t3.nano	0.03	5.06
t3.micro	0.06	5.09
t3.small	0.13	5.11
t3.medium	0.25	4.98
t3.large	0.51	5.11
t3.xlarge	1.02	5.11
t3.2xlarge	2.04	5.11
...		

Figure 5: Baseline Network Throughput by Instance Type [27]

With our average bandwidth around .9 Gbits/s, and our various use cases, it seems it would make sense to have a variety of instance sizes. In general however, we prescribe the t3.xlarge instance type to be able to support this average bandwidth and all the computing resources mentioned above, at the lowest cost to the user. In times of increased traffic, the T-family will also have the ability to burst to support even greater bandwidth. Lastly, it may be worthwhile to include some t3.2xlarge instances, especially if we find that the COR network is strained and needs to be able to better support the bandwidth requirements of onion streaming services.

3.4 Regions and Availability Zones

Now that we have evaluated t3.xlarge instances as the most practical instance type for a COR implementation, we shift our focus to analyzing the tradeoffs in the regions and availability zones

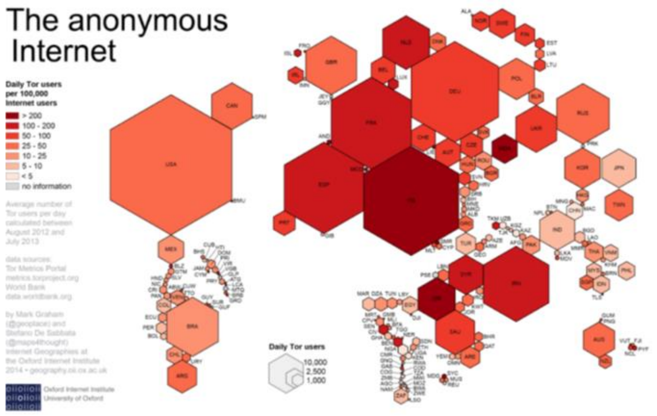


Figure 6: Cartogram of the Number of TOR Users per 100,000 Internet Users [21]

offered by AWS. There are approximately two million TOR users spanning the entire globe, subject to varied network connectivity and authoritarian censorship. Figure 6 [21] is a cartogram created by the Oxford Internet Institute, illustrating the number of TOR users per country, as well as the number of daily TOR users per 100,000 internet users. From the cartogram, we can see that the majority of

TOR's userbase is located within the United States and Europe, followed by middle eastern countries like Iran and Israel. A large percentage of internet users in Africa are also TOR users [24, 25].

The number of TOR users is likely influenced by the prevalence of relays within the same geographic region, as physical proximity is directly proportional to network latency—a connection between two nearby machines is much faster than a connection between two geographically separated machines. Figure 7 [14] shows a histogram of the locations of known TOR exit nodes based on their IP addresses from torproject.org. Assuming that this distribution is representative of the prevalence of guard and middle nodes, we can reason that users in the US or Europe experience much less latency within TOR connections than users in other countries.

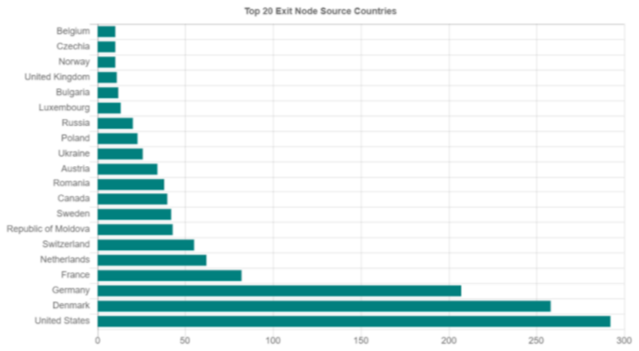


Figure 7: Number of TOR Exit Relays per Country [14]

In designing an ASP to provide relay access to COR users, the relative geographic locations of relays and users should be taken into account. AWS currently spans 77 availability zones within 24 geographic regions and plans to expand to include 12 additional zones in 4 additional geographic regions [4]. Figure 8 shows a map of the current and upcoming regions supported by AWS. Given the

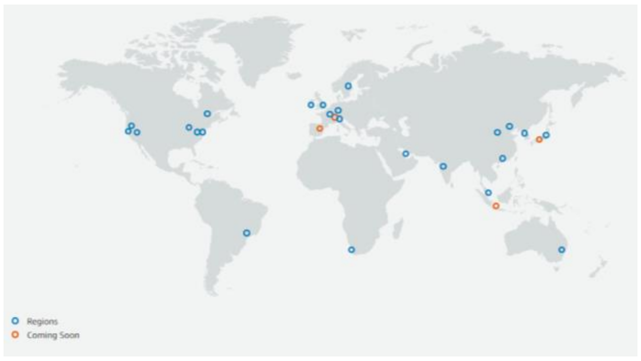


Figure 8: Geographical Visualization of Current and Upcoming Amazon Web Service Regions [4]

widespread availability of Amazon's data centers, COR can greatly improve the accessibility of onion routing networks across the world. By purchasing COR relays within a wide-ranging set of AWS availability zones, ASPs can cater to users across the world by providing users with access to geographically closeby relays. However, ASPs must consider the varying costs of hosting EC2 instances in certain regions [4]—yet another tradeoff between cost and the global availability of the COR network.

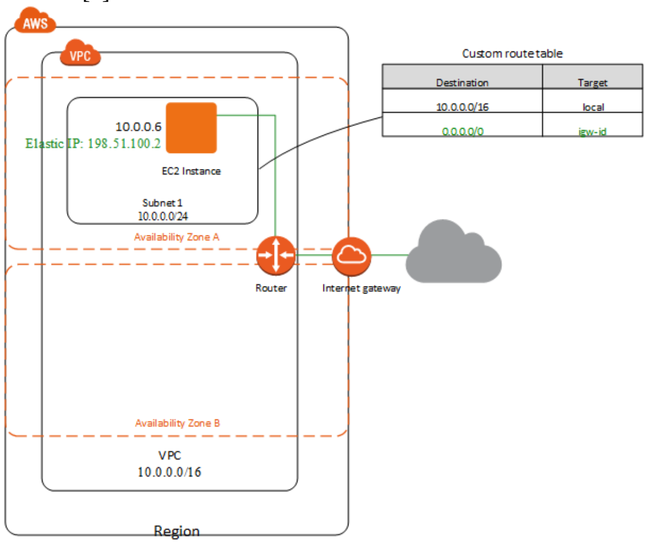
3.5 Network Topology

Jones et al. [16] recommends building a COR circuit where each COR tunnel has at least two relays within each datacenter it traverses. This property increases the difficulty for an adversary to monitor COR connections by requiring the adversary to eavesdrop on most ISP connections to each data center [16]. How should COR nodes within each datacenter be connected on the network? This presents a network topology design decision in the context of cloud computing.

All CHP have many of the same characteristics when it comes to networking like public and private subnets [11, 13]. We will focus on Amazon Web Services' Virtual Private Cloud (VPC) for this discussion. A VPC is a virtual network that is logically isolated from other networks in the AWS cloud. Within a VPC, a user can create multiple subnets, which are ranges of IP addresses. Public subnets are used for resources that must be connected to the internet, and private subnets are used for resources that are isolated from the internet [5].

We have chosen two of the most common and robust VPC implementations to explore for COR. We will analyze them across four different criteria: sender anonymity, receiver anonymity, performance, and security.

The first configuration, VPC with a single public subnet or VPC-public, is the simplest implementation of a possible COR VPC. As seen in Figure 9 below, it consists of a single public subnet, and an internet gateway to enable communication over the internet [6].

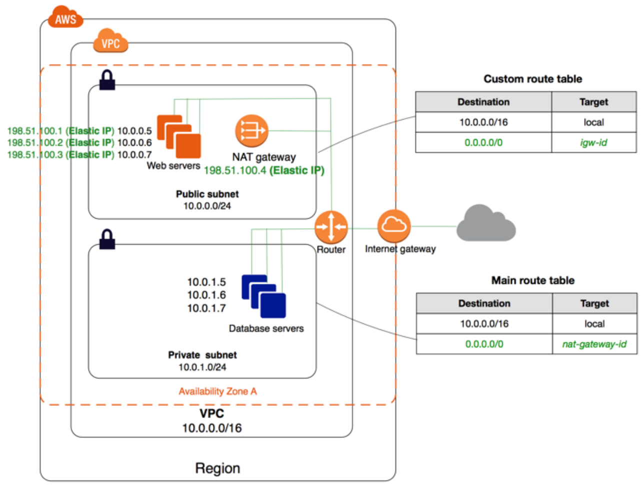


Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-id

Figure 9: VPC-public is a VPC with a single public subnet, and an internet gateway to enable communication over the

Elastic IP addresses can be assigned to Elastic Cloud Compute (EC2) instances that can be spun up and torn down within the VPC, eliminating wholesale blocking of COR. This will preserve, and even strengthen, sender anonymity compared to the traditional TOR implementation. By the same token, receiver anonymity will be strengthened compared to TOR since these ephemeral EC2 instance IP addresses will be more difficult for an adversary to monitor than static IP addresses like seen in TOR. Since this approach uses an internet gateway, performance will not be affected [22]. A downside of the VPC-public implementation is security. Since EC2 instances reside in a public subnet, all relays are accessible by the entirety of the internet.

The next configuration is a VPC with public and private subnets or VPC-public-private. As seen in Figure 10 below, the instances in the public subnet can send outbound traffic directly to the Internet, whereas the instances in the private subnet cannot. Instead, the instances in the private subnet can access the Internet by using a network address translation (NAT) gateway that resides in the public subnet [7]. This would allow TOR users to connect through EC2 instances in the public subnet then route traffic to one or many relays in its private subnet. From there, it can be sent to the end user



Custom route table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-id

Main route table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	nat-gateway-id

Figure 10: VPC-public-private is a VPC with where instances in the private subnet can access the Internet by using a network address translation (NAT) gateway that resides in the public subnet

or another CHP. This will provide similar sender anonymity like in the previous VPC-public implementation by leveraging floating EC2 IP addresses. However, VPC-public-private will have worse receiver anonymity since all traffic will be sent through the same NAT gateway, therefore having one IP address [8]. The NAT gateway could also impose limitations on performance at extreme traffic loads. When the bandwidth on a NAT exceeds 45 Gbps, additional overhead and cost will be required to distribute NATs across multiple subnets [9]. The VPC-public-private implementation provides better security than the VPC-public implementation since some, but not all relays are accessible from the public internet.

The comparison between the VPC-public and VPC-public-private implementations provides insights into some of the tradeoffs required when following Jones et al's recommendation for building a COR circuit.

We recommend using the VPC-public-private implementation because when prioritizing security this implementation presents the best compromise between sender anonymity, receiver anonymity, performance, and security. The VPC-public-private implementation is similar to the VPC-public implementation since both increase sender anonymity compared to a traditional TOR implementation. Although the VPC-public-private implementation does not augment receiver anonymity like VPC-public, it will perform similar to a traditional TOR implementation. Furthermore, the VPC-public-private will require additional overhead when network bandwidth exceeds 45 Gbps. However, this is a tradeoff we are willing to make because it will increase security of the overall COR network by allowing CHP to utilize private subnets in the VPC-public-private implementation.

4 Results

Based on our in-depth analysis of the tradeoffs of network topology, instance purchasing options, instance family types, typical applications on TOR and their required bandwidth, and regions and availability zones we now prescribe the most optimal configuration based on latency, throughput, monetary cost per user, usability, and of course the preservation of security from local and global network adversaries.

We recommend a network topology implementation that consists of a VPC with public and private subnets where the private subnet can access the Internet by using a network address NAT gateway that resides in the public subnet. When prioritizing security this implementation presents the best compromise between sender anonymity, receiver anonymity, performance, and security.

As for reservation type, spot instances, while available and of large enough size to support the 0.9 Gbit/s bandwidth requirement, are the best choice to make COR as cheap as possible for end users. If a relay is about to be turned off by the CHP, the CHP will give a 5 minute warning to the ASP. This 5 minutes provides adequate time for the ASP to request sufficient on-demand resources to replace these spot instances in the network and for the ASP to advertise these new routes. On-demand instances can supplement the supply of spot instances to ensure adequate coverage among all regions and global availability.

When considering instance family and size for use as COR relays, we consider memory and bandwidth restraints for guard, middle, and exit relays. Given the minimum requirements of 1 GB RAM for non-exit nodes, 1.5 GB RAM for exit nodes, and 0.9 Gbit/s bandwidth for all nodes, an ASP can minimize costs by leveraging the T-family of EC2 instances, with a minimum size requirement of ‘xlarge’. An ASP should consider offering more performant instances to serve more bandwidth-heavy services like streaming services.

5 Conclusion and Future Work

We sincerely hope that our work will add value to the fairly novel design space of COR, and that once implemented, COR will better defend the Internet freedom and privacy of users over the existing TOR. We leave quantitative experiments based on our recommendations for future work. It would be useful to explore how the additional overhead of adding NAT gateways when traffic increases over 45 Gbps affects performance in the VPC-public-private implementation. Furthermore, measuring how our choice of a mixture of spot and on-demand T-family instances performs in real-world production environments against latency, availability, and throughput as compared to current TOR relays.

REFERENCES

[1] Amazon Web Services. 2020. Instance purchasing options. (2020). Retrieved November 7, 2020 from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-purchasing-options.html

[2] Amazon Web Services. 2020. Amazon EC2 Instance Types. (2020). Retrieved from https://aws.amazon.com/ec2/instance-types/.

[3] Amazon Web Services. 2020. Amazon EC2 On-Demand Pricing. (2020). Retrieved from https://aws.amazon.com/ec2/pricing/on-demand/.

[4] Amazon Web Services. 2020. Global Infrastructure. (2020). Retrieved from https://aws.amazon.com/about-aws/global-infrastructure/.

[5] Amazon Web Services. 2020. What is Amazon VPC?. (2020). Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

[6] Amazon Web Services. 2020. VPC with a single public subnet. (2020). Retrieved November 8, 2020 from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario1.html

[7] Amazon Web Services. 2020. VPC with public and private subnets (NAT). (2020). Retrieved November 8, 2020 from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario2.html

[8] Amazon Web Services. 2020. NAT gateways. (2020). Retrieved November 8, 2020 from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

[9] Amazon Web Services. 2020. Amazon VPC pricing. (2020). Retrieved November 8, 2020 from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

[10] Gavin Cahill. 2020. Definitive Guide to AWS EC2 Pricing and How to Control Costs. (October 2020). Retrieved November 7, 2020 from https://www.apptio.com/blog/guide-to-aws-ec2-costs/

[11] Fidelis Ekezue. 2017. Differentiating between Azure Virtual Network (VNet) and AWS Virtual Private Cloud (VPC). (September 2017). Retrieved November 8, 2020 from https://devblogs.microsoft.com/premier-developer/differentiating-between-azure-virtual-network-vnet-and-aws-virtual-private-cloud-vpc/

[12] Sylvia Engdahl. 2018. New Research From TSO Logic Shows AWS Costs Get Lower Every Year. (2018). Retrieved November 7, 2020 from https://aws.amazon.com/blogs/apn/new-research-from-tso-logic-shows-aws-costs-get-lower-every-year/

[13] Google Cloud. 2020. Patterns for connecting other cloud service providers with Google Cloud. (2020). Retrieved November 8, 2020 from https://cloud.google.com/solutions/patterns-for-connecting-other-csps-with-gcp

[14] Hacker Target. Tor Exit Nodes Located and Mapperd. Retrieved from https://hackertarget.com/tor-exit-node-visualization/.

[15] Intel. Advanced Encryption Standard New Instructions. Retrieved from https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html.

[16] Nicholas Jones, Matvey Arye, Jacopo Cesareo, and Michael J. Freedman. Aug. 2011. Hiding amongst the clouds: A proposal for cloud-based onion routing. In FOCI.

[17] Risto Laurikainen. 2010. Secure and anonymous communication in the cloud. Aalto University School of Science and Technology—Department of Computer Science and Engineering, Tech. Rep. TKK-CSE-B10, 1-5.

[18] Galia Novakova Nedeltcheva, Elior Vila, and Marina Marinova. 2019. The onion router: Is the onion network suitable for cloud technologies. In Smart Technologies and Innovation for a Sustainable Future, 389-398.

[19] ORNetStats. 2020. Onion Routing Network Statistics. (2020). Retrieved from https://nusenu.github.io/OrNetStats/#os-distribution-relays.

[20] Dr. Gareth Owen. 2015. Tor: Hidden Services and Deanonymisation. Retrieved November 7, 2020 from https://www.youtube.com/watch?v=-oTEoLB-ses&t=1998

[21] Oxford Internet Institute. 2014. The Anonymous Internet. (2014). Retrieved from http://geography.oii.ox.ac.uk/the-anonymous-internet/.

[22] Ashish Patel. 2019. AWS — Difference between Internet gateway and NAT gateway. (May 2019). Retrieved November 8, 2020 from

https://medium.com/awesome-cloud/aws-vpc-difference-between-internet-gateway-and-nat-gateway-c9177e710af6
[23] TorCommunity. Relay Requirements. Retrieved from https://community.torproject.org/relay/relays-requirements/.
[24] TorMetrics. 2020. Top-10 countries by relay users. (2020). Retrieved from https://metrics.torproject.org/userstats-relay-table.html.
[25] TorMetrics. 2020. Top-10 countries by bridge users. (2020). Retrieved from https://metrics.torproject.org/userstats-bridge-table.html.
[26] The Tor Project. 2018. Tor Metrics: Traffic. (2018). Retrieved November 7, 2020 from https://metrics.torproject.org/advbwdist-perc.html
[27] Andreas Wittig. 2018. EC2 Network Performance Cheat Sheet. (January 2018). Retrieved November 7, 2020 from https://cloudonaut.io/ec2-network-performance-cheat-sheet/