Hiding Amongst the Clouds: A Proposal for Cloud-based Onion Routing

Nicholas Jones, Matvey Arye, Jacopo Cesareo, and Michael J. Freedman
Princeton University

Abstract

Internet censorship and surveillance have made anonymity tools increasingly critical for free and open Internet access. Tor, and its associated ecosystem of volunteer traffic relays, provides one of the most secure and widely-available means for achieving Internet anonymity today. Unfortunately, Tor has limitations, including poor performance, inadequate capacity, and a susceptibility to wholesale blocking. Rather than utilizing a large number of volunteers (as Tor does), we propose moving onion-routing services to the "cloud" to leverage the large capacities, robust connectivity, and economies of scale inherent to commercial datacenters. This paper describes Cloud-based Onion Routing (COR), which builds onion-routed tunnels over multiple anonymity service providers and through multiple cloud hosting providers, dividing trust while forcing censors to incur large collateral damage. We discuss the new security policies and mechanisms needed for such a provider-based ecosystem, and present some preliminary benchmarks. At today's prices, a user could gain fast, anonymous network access through COR for only pennies per day.

1 Introduction

Tor is one of the most popular tools for accessing the Internet anonymously. Tor operates by tunneling a user's traffic through a series of relays (proxies), allowing such traffic to appear to originate from its last relay, rather than the user. Tor relays are hosted by volunteers all over the world, which is beneficial in providing jurisdictional arbitrage for relayed traffic. Yet, its reliance on volunteers also results in adverse performance: Many Tor relays offer only consumer-grade ISP connections, which often have poor "last mile" latency and bandwidth capacity. Such clients also often have highly asymmetric network connectivity, which is ill-suited for a relay's equal use of upstream and downstream bandwidth. Additionally, Tor nodes are advertised publicly via Tor directory servers. While helping to prevent partitioning attacks [6], this openness makes Tor vulnerable to censorship: Any censoring government can easily enumerate the IP addresses of all public Tor relays and block them.

Cloud infrastructure is everything that Tor is not: there are a *smaller* number of providers, yet they administer a much *larger* number of high-quality, high-bandwidth nodes. While fewer in number, these providers are still spread over multiple administrative and jurisdictional boundaries. Further, cloud infrastructure is *elastic*: One can incrementally add (or remove) relays to a cloud, simply by spinning up new virtual machine (VM) instances in the cloud. This can be particularly effective given the typical diurnal nature of client demand. This elasticity also inhibits wholesale blocking. While Tor relays commonly use static addresses, clouds allow one to rotate between IP addresses quickly (either by retiring and spinning up new VMs, or having existing VMs release and acquire new IPs). A censor is thus left the option of either blocking all IP prefixes used by the cloud provider—and causing large *collateral damage*—or allowing the traffic to flow untrammeled.

Yet we are faced with a security dilemma: Does adopting a cloud deployment threaten the very anonymity we seek? There are already numerous privately-hosted anonymity solutions (proxies like anonymizer.com, private VPNs, etc.). These existing solutions require the client to fully trust their provider, however. Instead, we propose a hybrid solution: build a high-quality Tor-based network on multiple cloud hosting providers. Much like Tor has a *separation of trust* between its unknown volunteers, we base security on the assumption that multiple autonomous and known entities involved in an onion-routed network will not collude.

Fundamentally, COR does not change the basic Tor protocol. Instead, it proposes a means to establish Tor-like tunneling in a more market-friendly setting of anonymity service and cloud hosting providers. Of course, this raises its own technical challenges and security concerns. These challenges include how end-users pay for (or be freely given) access to relays while preserving privacy, and how clients can discover relays without being vulnerable to partitioning attacks.

This paper presents an overview of the COR design and explores the design space of building anonymity services using cloud providers. We discuss the new secu-

rity challenges that arise from this setting, as well as the new opportunities for scalability, robustness, and performance. We conclude with preliminary benchmarks from our early COR prototype.

2 System Overview

2.1 High-Level Design

To establish an anonymous connection in COR, an end-user iteratively builds an encrypted tunnel through a circuit of relay nodes, much like in Tor. To create an incentive for operating anonymizing relays, however, users have to "pay" for the right to use COR relays. This raises a security challenge: If COR users were to provide payment directly to cloud hosting providers (CHPs), their billing information could be used to de-anonymize their Internet access (*e.g.*, the entry CHP could correlate billing information to client IP, while the exit CHP could correlate billing information to request plaintext).

We overcome this problem by using a layer of indirection. COR separates the role of operating anonymizing relays (by so-called anonymity service providers, or ASPs) from the actual CHPs that manage the infrastructure. These ASPs rent VMs, run anonymizing relays in these VMs, and accept cryptographic tokens from connecting users in exchange for relaying their traffic. These tokens have the cryptographic property that it is impossible to link the purchase of a token with the redemption of the token, preventing ASPs from determining which user redeemed a particular token (as described in section 3.2). We envision that ASPs could even federate to accept tokens issued by other ASPs. While some prior work has suggested the use of cloud VMs as Tor relays [8], unlike COR, it did not support the creation of an economic ecosystem that would allow users to purchase cloud-based anonymity services securely.

COR's use is characterized by two phases. First, clients engage in the process of obtaining tokens and learning the set of relays in the federation. Second, clients build an onion-routed circuit by redeeming the tokens at the relays of their choice; they then use this circuit for anonymous communication. This separation is analogous to the control/data plane separation of other capabilities-based systems that seek to protect against denial-of-service attacks [11, 12], in which the data plane is protected by a capability, while the control plane needs to be protected by other means (although COR's focus is on anonymity, rather than DoS protection).

Given the phases' different security concerns, COR is composed of two separate relay networks conceptually:

1. The **bootstrapping network** allows users to preserve anonymity when starting to use COR. A user can use this network to ensure IP privacy while acquiring tokens, obtaining directory server information, and establishing an initial circuit. Since a user does not initially have tokens, the bootstrapping network does not require tokens to use its relays. To prevent abuse, however, it is limited in that it can only be used to access COR directory and token servers, and not the wider Internet.

2. The **data network** is a high-bandwidth, low-latency network through which users are able to anonymously access the Internet. Having acquired tokens and a list of relays during the bootstrapping phase, a client can now build an onion-routed circuit. To extend a circuit to a new relay, the client provides a valid token to that relay, which grants it temporary access (typically metered by consumed bandwidth). The user repeats this process multiple times to build the full circuit.

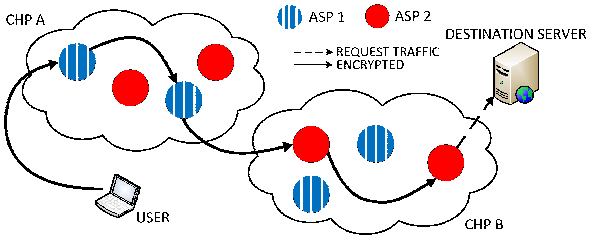

Figure 1: **COR system overview. Users communicate over an onion-routed tunnel built over servers operated by multiple Anonymity Service Providers, which are located in the datacenters of multiple Cloud Hosting Providers.**

We illustrate a COR data tunnel in Figure 1, where the user establishes a circuit through relays managed by ASP 1 and 2, which includes traversing clouds operated by CHP A and B. The user's data is hop-by-hop onion encrypted along the circuit exactly like in Tor, serving to anonymize the user from the relays he uses, the clouds he traverses, and other network eavesdroppers he confronts.

2.2 System Trust and Threat Model

COR's trust model is necessarily different from Tor's because it introduces new roles and relationships into the system. Within Tor, there are only two roles: end-users seeking anonymity and relay operators that provide it.[1] In COR, two administratively-distinct parties have access to a relay: the ASP that directly operates it, and the CHP that has administrative access to the physical machine (and the virtual machine's hypervisor). Thus, we need to discuss the threats posed by both ASPs and CHPs, in addition to malicious users and censors.

The threat model for ASPs and CHPs is very similar. While the tunnel's security relies on the fact that some of the involved ASPs/CHPs do not collude, individual malicious ASPs or CHPs may keep detailed logs, packet traces, or otherwise attempt to de-anonymize users. As we discuss in Section 2.4, due to the risks of traffic anal-

[1] We do not focus on the different roles that "guard" or "exit" nodes play Tor, although we note that first and last COR hops can play similar roles. We do expect all ASPs to run exit nodes, unlike in Tor.

ysis, the same ASP should not appear in multiple, non-contiguous places within a circuit. Similarly, because CHPs have administrative control over ASPs' virtual machines, a user's circuit should not use relays that are all controlled by the same CHP.

End users have limited ability to attack COR. Users can attempt to perform denial-of-service attacks on the network. Malicious users can also attempt to alter the load characteristics of relays to perform side-channel attacks. Due to clouds' traffic volumes and ASPs' ability to spin up new instances when relays become loaded, we believe these attacks can be made ineffective.

2.3 Censorship Resistance

As in Tor, COR should protect against network level censors and monitoring eavesdroppers. These adversaries may be government agencies, ISP monitors, or corporations wishing to monitor or block traffic. We assume that such adversaries can monitor an end-user's traffic and have the ability to block traffic to specific addresses. However, there are limits to the power of any such adversary. Traffic monitoring, for example, cannot take place outside of an organization's jurisdiction. We anticipate that datacenters in COR will have a large number of incoming and outgoing connections from multiple ISPs [13]. This makes timing analysis and measurement significantly more difficult than in Tor, due to a cloud datacenter's number of connections, traffic volume, heterogeneous use, and asymmetric routing.

In addition to better network connectivity and dynamic scaling, cloud infrastructure also inhibits blocking. Some clouds allow virtual machine instances to switch IP addresses quickly (*e.g.*, using DHCP and gratuitous ARPs), while IP addresses can be rotated through in others by allocating new instances. In both cases, blocked IP addresses can be retired and new ones adopted. A censor is thus left with two options: block all IP prefixes used by the cloud provider, or otherwise allow the traffic to flow mostly untrammeled. This becomes a problem of *collateral damage*: Amazon EC2, for example, hosted over a million instances that share common IP prefixes in 2010 [9]. Admittedly, a determined adversary might be able to dynamically track IP addresses within the cloud and "whitelist" certain services. However, this would at be be a cat and mouse game of blocking, which would need to occur on all cloud services simultaneously. With the exception of Egypt's "disconnecting" itself from the wider Internet in January 2011—a practice reversed only a few days later—censors appear hesitant to enact widespread Internet blocking.

2.4 Building a COR Circuit

In Tor, the relays which a user chooses are picked mostly at random. However, due to the fact that there exist a limited number of ASPs and CHPs within COR, additional care is needed when constructing a COR circuit. The following properties describe an ideal COR circuit.

1. ***A COR tunnel has at least two relays within each datacenter it traverses.*** This property increases the difficulty with which COR connections can be monitored by an adversary with access to the traffic entering and leaving, but not internal to, the datacenter. If data leaves the datacenter from a different node than from which it entered, then such an adversary is relegated to using inefficient side-channels to correlate the traffic. Since almost all large datacenters are multi-homed [13], it is also likely that traffic will enter on a different ISP than the one from which it exits (unlike in Tor). Thus, for an adversary to reliably monitor COR traffic, he would have to eavesdrop on most ISP connections to each datacenter.

2. ***The entry and exit ASPs of a COR tunnel differ.*** This property ensures that any single compromised ASP will know limited information about a COR user's tunnel. If the tunnel's first-hop ASP logs information, it can record the tunnels' originating IP address, but not the corresponding request plaintext. Similarly, the exit ASP can learn the resource being requested, but cannot link the request to an originating client IP address.

3. ***The entry and exit CHPs of a COR tunnel differ.*** This property, analogous to the previous, protects against a compromised CHP rather than a compromised ASP.

4. ***Relays in a COR tunnel belonging to the same ASP are not separated by a single hop.*** When a single ASP's relays surround the relay of another ASP within a circuit, the second ASP's relay adds no additional anonymity. If the first ASP is malicious, it can relatively easily correlate the traffic entering and leaving the second ASP's node via the relay's IP address, thus defeating the purpose of the extra hop.

2.5 Market costs for COR

In order to motivate the design of the COR system, it is worthwhile to understand its resource costs compared to a volunteer network like Tor. By examining Tor's metrics page, we estimate that an upper bound for the daily-averaged bandwidth consumed by Tor relays during the spring of 2011 was approximately 900 MB/s. Given three-hop Tor circuits, with each relay shuttling the same encrypted bytes from upstream to downstream, this total translates to an end-user bandwidth demand of 150 MB/s, or approximately 376 TB/month. From this, we now calculate the approximate cost of running a COR network with the same bandwidth. For the sake of these calculations, we will use Amazon EC2's bandwidth pricing. As of July 2011, Amazon charges nothing for downstream traffic and uses a sliding scale for upstream traffic; intra-cloud traffic is free. The first 10TB are billed at $0.12/GB, the next 40TB at $0.09/GB, the next 100TB

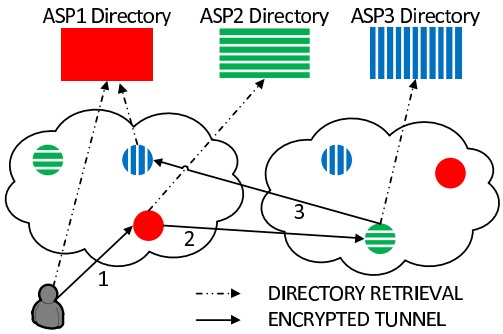

Figure 2: **Directory retrieval through the COR bootstrapping network. Relay colors denote the operating ASP.**

at $0.07/GB, and the next 350TB at $0.05/GB. A typical COR circuit consists of two clouds, each with two nodes, so there are two upstream connections and two downstream connections in our circuit. Each connection experiences 150 MB/s of traffic, so we can estimate the approximate monthly cost of COR as $61,200 for its 376 TB. This cost would be spread across all ASPs.

Further, bandwidth costs continue to decrease. Since its launch in 2008, Amazon's bandwidth costs have decreased by over 70% [1]. Finally, cheaper hosting services are also available, *e.g.*, some price 10 TB/month at under $100, leading to per-month costs of $37,000. That said, smaller hosting providers might not provide the same elasticity and threat of collateral damage as their more well-known counterparts.

3 System Design

This section elaborates on some of COR's design details. First, we discuss the process through which users contact ASP directory servers to discover relays. Next, we discuss the properties of COR tokens and their distribution methods. Finally, we discuss the authentication and token redemption mechanisms of COR relays.

3.1 Retrieving the COR Directory

Before a user can build a COR circuit across multiple ASPs' relays, a user must discover potential relays from the ASPs' directories. Directories are responsible for tracking the available COR nodes for a given ASP. In Tor, directories are public, and any user can download the entire directory at any time. This makes the nodes returned by these directories vulnerable to IP-address-based blocking by censors.[2]

[2]Tor has introduced the notion of more trusted private relays that are not listed in public directories. Tor still faces the problem of distributing these lists to "desired" users, yet not to censors, who actively attempt to discover these relays' IP addresses. Given that the set of Tor private relays is still relatively small and fixed—and that distinguishing between unauthenticated users and censors is difficult—most private relays have been blocked by some censoring countries [7].

When a user retrieves the COR directory, they do not receive the entire list of available nodes, but only a subset of the available nodes. Unfortunately, this design creates new vulnerabilities. Consider the scenario of a malicious directory server. Since COR directories only return a small subset of the total number of nodes, a malicious directory server could target an individual user (or, more specifically, the user's IP address) and give that user a uniquely-identifying list of relay nodes. To prevent this kind of partitioning attack, directory retrieval within COR always occurs through a COR circuit that hides the user's true identity by anonymizing access. Initially, when a user does not have an established COR circuit in the data network, the user can use the bootstrapping network to create an anonymizing circuit and retrieve the directory.

A bootstrapping fetch works as follows: Any user can contact a directory and ask for a bootstrapping relay without supplying a token. The user adds the given bootstrap node to his circuit, and then contacts a different directory through this circuit. This process is repeated until the user has fully constructed his bootstrapping circuit. The user thus incrementally builds his circuit. Once the circuit is complete, the user is able to purchase tokens or retrieve a directory for the data network.

3.2 COR Tokens

COR tokens provide the means for a user to gain access to a relay for some specified duration and/or transfer size. This is implemented with Chaum's blinded signature scheme [4], where a user, when purchasing or otherwise acquiring a token, sends a blinded random nonce to the ASP. The ASP then replies with a signature of the random nonce without ever learning its value. When redeeming the token, the user sends the signed random nonce to the ASP, which upon verifying the signature (and the fact that the nonce was never used before), authorizes the token and adds the nonce to the list of used nonces. Since each blinded signature can only produce a valid signature for one nonce, and the ASP keeps a list of used nonces, it is assured that a token can only be used once. The user is assured that privacy is preserved because it is computationally hard for the ASP to correlate the blinded nonce it learned when the user acquired the token with the signed nonce sent to it during token redemption. Previous proposals like XPay [5] and Par [2] offer more complex solutions than those needed by COR. XPay deals with micropayments, while Par assumes a single central bank, neither of which applies to COR. Bitcoin is not appropriate for use as a token because all transactions are logged and stored within the Bitcoin network [3]. However, Bitcoin could be used as a currency for purchasing tokens, much like any other currency that an ASP chooses to accept.

Circuit
R0 → R1 → US1 → US2
US1 → US2 → R0 → R1
R0 → R1 → EU1 → EU2
EU1 → EU2 → R0 → R1
US1 → US2 → R0 → R1 → EU1 → EU2

Table 1: **Evaluated COR circuits traversing Rackspace (R), Amazon EC2 East (US), and EC2 Europe (EU).**

Distributing COR tokens while preserving anonymity presents a challenge in practice. The original work on e-cash assumed the use of anonymous channels, the very thing we are trying to build! Using whatever criteria the ASP deems necessary, tokens may be distributed to users. However, most ASPs will want to authenticate the user in some way before granting a user a token. For example, ASPs may want to authenticate the user's affiliation with an organization for which they seek to provide free access (using whatever means desired). Alternatively, they may want to ensure the user provided payment. In any case, if the IP address used during token acquisition is the same IP that will be used when accessing COR, the ASP can de-anonymize the user when the token is redeemed. We can prevent this attack by making sure that all access to the token server is done through a COR circuit. If a user does not have access to COR relays within the data network already—*e.g.*, he is connecting to COR for the first time—he can access ASPs via the bootstrapping network.

3.3 Authenticating with Relays

When a user attempts to connect to a COR relay, the user must present a COR token to the relay. The relay then immediately contacts the ASP which issued the token to check its validity. If the token is accepted, the user gains access to that relay according to the terms stipulated by the ASP, with the relay measuring the connection's aggregate resource use. When the connection is about to expire or exceeds its approved usage, the user can redeem an additional token to keep the circuit alive.

4 Evaluation

Tor performance is one of the major factors inspiring our work on COR. Thus, our preliminary evaluation considers the performance of COR vs. Tor. We used two experiments for this comparison: (i) downloading individual files (via TorPerf) and (ii) downloading entire web sites. We also evaluate how many concurrent users a single cloud node can support, to both offer deployment guidance and to justify the assertion that operational costs are bandwidth dominated.

Our COR prototype is a modified version of Tor 0.2.1.29, allowing us to easily specify arbitrary circuits through the cloud. For this initial evaluation, we have not yet implemented cryptographic token exchange within our prototype. The following results show benchmarks measured after the connection has been established. We anticipate that the COR token exchange will add only modest latency during connection setup.

4.1 Individual File Downloads

To evaluate the performance of COR when downloading individual files, we use the Tor-supplied TorPerf [10] measurement tool. Our COR experiments used five COR circuits which were statically built across US and European cloud datacenters, as described in Table 1, as well as five Tor circuits, randomly chosen by Tor. For each circuit, we started a different client (located on a Princeton server) and downloaded 3 files of size 50 KB, 1 MB, and 5 MB, repeating each download 100 times. After each run, we shut down the client before restarting the experiment on a new circuit. We ensured that Tor did not switch circuits mid-run by setting its configuration parameter *MaxCircuitDirtiness* to 60 minutes. The performance results are shown in Figure 3(left). We can see that the average COR performance is superior to TOR for all files and that the best COR circuit of the five has a median performance that is 7.6× faster than that of Tor.

4.2 Downloading Web pages

We also evaluate the time it took to download an entire web page (using `wget -p` to ensure that the page and all its associated content are downloaded). We downloaded the home pages of the top 10 Alexa.com domains,[3] using Tor, COR, and a direct Internet connection. The COR downloads were performed over the same five circuits given in Table 1, and Tor circuits were restarted at the same frequency as COR circuits. Each home page download was performed 10 times consecutively per circuit. For the COR downloads, we tested both a COR relay that was serving 50 simultaneous connections and a COR relay that was serving a single client. Because we do not have data describing the load of deployed Tor relays, we sought to evaluate Tor against both highly loaded and unloaded COR instances. It is important to note, however, that the elastic resource scaling of clouds make it possible to bring up new COR instances under load. That said, Figure 3(center) shows that COR is several times faster than Tor for all sites, even under load, although it is still slower than direct access for most sites.

4.3 Concurrent Users

Finally, we evaluate the number of concurrent users that a single relay in the cloud can support while still offering acceptable throughput per client. We evaluated

[3] We omit results from qq.com due to space constraints and because its poor performance over all trials do not allow for easy illustration.

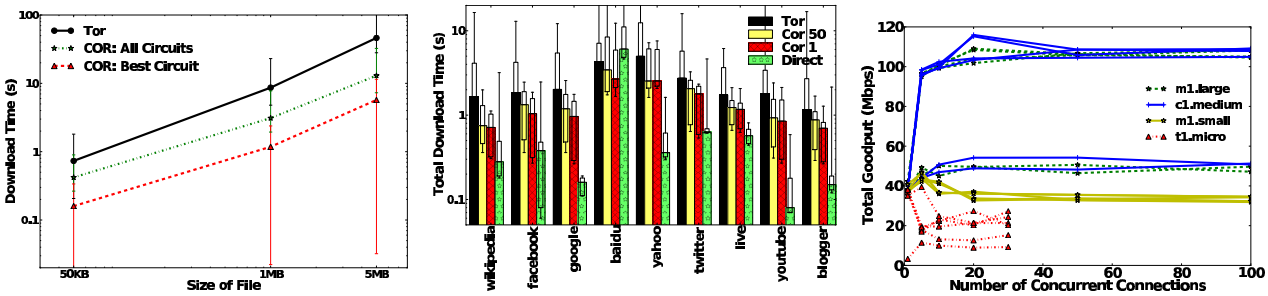

Figure 3: **The left plot shows median file download time using TorPerf, measured between the first and last byte received of the HTTP response. Error bars illustrate quartile performance. The center plot shows median download times for a selection of popular websites. Error bars show 90th and 95th percentile times. The right plot shows aggregate COR bandwith through various types of Amazon EC2 instances.**

different "size" nodes from Amazon EC2 for this experiment, which corresponded to differing amounts of CPU, memory, and I/O resources per size. (In decreasing price are *standard large*, *high cpu medium*, *standard small*, and *micro* nodes.) Each circuit was built using two nodes of the same size and located in the same datacenter. The experiment consisted of using TorPerf to download a 50 MB file from Amazon's S3 storage service and evaluating the bandwidth of the transfer. The results are shown in Figure 3(right). Two of the larger node types ("m1.large" and "c1.medium") can easily handle more than 100 concurrent users, while the cheap "t1.micro" node struggles to support ten users. One interesting finding is that the larger nodes achieved a maximum of either 50 Mbps or 100+ Mbps. We discovered that even among nodes of the same type, the maximum bandwidth differed depending upon which nodes we were assigned. We suspect that some nodes within Amazon EC2 may have slightly faster network infrastructure than others. Unfortunately, Amazon does not guarantee specific upstream or downstream bandwidths, although an ASP could test assigned nodes for particular characteristics.

In summary, an EC2 node costing as low as 17¢ per hour, plus bandwidth charges, can relay approximately 110 Mbps. This can provide up to 100 concurrent users an average of 1 Mbps of bandwidth each.

5 Conclusion

COR explores a new direction for low-latency onion-routing systems: leverage the large capacities, robust connectivity, and economies of scale inherent to elastic cloud infrastructures. But COR still avoids trusting any single entity, as COR users build circuits over multiple ASPs and through multiple CHPs. In doing so, COR creates a potential marketplace or exchange for anonymity services: allowing multiple, independent parties to coexist, with each facing little to no *capital* costs. COR also introduces new protections against blocking, confronting censors with tough choices: They must either allow access, engage in a cat-and-mouse game against relays' transient addresses, or block cloud providers' entire prefixes and cause significant collateral damage. COR does not solve the fundamental bootstrapping problem inherent to many anonymity systems – users' initial connections to the COR bootstrapping network are vulnerable to the same attacks as traditional Tor connections. We leave this to future work. Preliminary evaluations demonstrate that COR can be both efficient, high performing, and cost effective. Our ongoing work seeks to further develop COR's mechanisms for token acquisition and exchange, in order to establish an efficient market for Internet-scale anonymity services.

References

[1] Amazon AWS Forums. `https://forums.aws.amazon.com/ann.jspa?annID=196`.

[2] E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. Bellovin. PAR: Payment for anonymous routing. In *Proc. PET*, 2008.

[3] Bitcoin Anonymity. `https://en.bitcoin.it/wiki/Anonymity`.

[4] D. Chaum. Blind signatures for untraceable payments. In *Proc. CRYPTO*, 1982.

[5] Y. Chen, R. Sion, and B. Carbunar. XPay: Practical anonymous payments for tor routing and other networked services. In *Proc. WPES*, 2009.

[6] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III anonymous remailer protocol. In *Proc. IEEE Security and Privacy*, 2003.

[7] R. Dingledine. Private communication, 2010.

[8] R. Mortier, A. Madhavapeddy, T. Hong, D. Murray, and M. Schwarzkopf. Using dust clouds to enhance anonymous communication. In *Proc. IWSP*, 2010.

[9] Recounting EC2 One Year Later. `http://www.jackofallclouds.com/2010/12/recounting-ec2/`.

[10] TorPerf. `https://metrics.torproject.org/tools.html`.

[11] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate DDoS flooding attacks. In *Proc. IEEE Security and Privacy*, 2004.

[12] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proc. SIGCOMM*, 2005.

[13] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *Proc. NSDI*, 2010.