Hiding Amongst the Clouds:

A Proposal for Cloud-based Onion Routing

Nicholas A. Jones

A Thesis

Presented to the Faculty

of Princeton University

in Candidacy for the Degree

of Master of Science in Engineering

Recommended for Acceptance

by the Department of Computer Science

Adviser: Michael J. Freedman

June 2012

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 benchmarks from our COR implementation. At today's prices, a user could gain fast, anonymous network access through COR for only pennies per day.

Acknowledgments

I would like to thank my adviser, Michael Freedman for his advice and support during my time at Princeton. This research would not have been possible without his input and guidance (and occasional prodding). I would like to thank and acknowledge Matvey Arye and Jacopo Cesareo for their collaboration on an early version of this work. Finally, I'd like to thank the members of the SNS Research Group and the Center for Information Technology Policy for their feedback, criticisms, and suggestions during the course of this research.

This thesis is dedicated to my parents, for all their love and support through the years.

Contents

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 [11], 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 increases resistance to 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 [4], HotSpotVPN [16], etc.). However, these existing solutions require the client to fully trust their provider. 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 security challenges that arise from this setting, as well as the new opportunities for scalability, robustness, and performance. We present benchmarks from a working COR prototype, and we conclude with a discussion of other models for coupling access control with anonymous browsing.

2 System Overview

In this section, we present an overview of the COR system. We discuss our threat model, geographic diversity in the Tor network, new techniques for censorship resistance, and the economic model upon which COR is based.

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 task of operating relays into two separate roles. In COR, **cloud hosting providers** (CHPs) manage the infrastructure, but provide no special accommodation for anonymous browsing. Conversely, **anonymity service providers** (ASPs) rent VMs, run anonymizing relays in these VMs, and accept crypto-

graphic 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 3.3). We envision that ASPs could even federate to accept tokens issued by other ASPs.

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 [26, 27], 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. In practice, the existing Tor network could serve this purpose, or regular COR relays could service this traffic, albeit with a highly throttled connection.

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.

We illustrate a COR data tunnel in Figure 1, where the user establishes a hypothetical circuit through relays managed by the Electronic Frontier Foundation and "Moneybags Corporation" (an imaginary for profit company), which includes traversing clouds operated by Amazon and Rackspace. The user's data is hop-by-hop onion encrypted along the circuit exactly like in Tor,

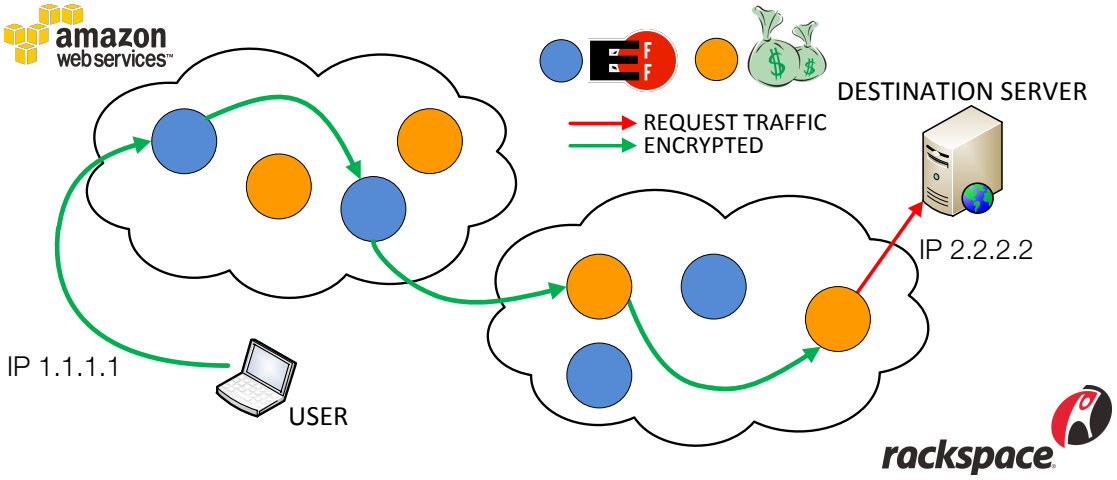

Figure 1: COR system overview. Users communicate over an onion-routed tunnel built over servers operated by multiple Anonymity Service Providers, which are located in the datacenters of multiple Cloud Hosting Providers. In this figure, the Electronic Frontier Foundation and a hypothetical "Moneybags Corporation" are shown as example ASPs. Amazon and Rackspace are used as example CHPs. These entities are used as examples only.

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 most are likely honest and, in fact, 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 3.1, due to the risks of traffic analysis, the same ASP

[1]We do not focus on the different roles that "guard" or "exit" nodes play in 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.

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. This can limit system availability, but does not compromise users' anonymity. 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. Overall, we believe that COR does not introduce new end-user attack vectors compared to Tor.

2.3 Geographic Diversity

Given COR's cloud based deployment, there are naturally fewer total administrative entities in the COR system. However, unlike in Tor, ASPs and CHPs have direct incentives to guard the privacy of their users. A primary reason for this is that COR users pay ASPs and CHPs for their service, so they are incentivized to retain those customers. Hypothetically, if a COR user were to be de-anonymized, the negative press from such an action would very likely have negative repercussions for the ASPs and CHPs involved. Additionally, ASPs and CHPs have significant infrastructure they must maintain in order to participate in the COR system, unlike in Tor. For example, a CHP must setup a datacenter and establish significant physical infrastructure. Similarly, an ASP must setup a payment infrastructure and establish a user base for its service. From this, we argue that although there are fewer total entities in the COR system, individual entities within the system are more highly incentivized to be trustworthy than individual entities within Tor.

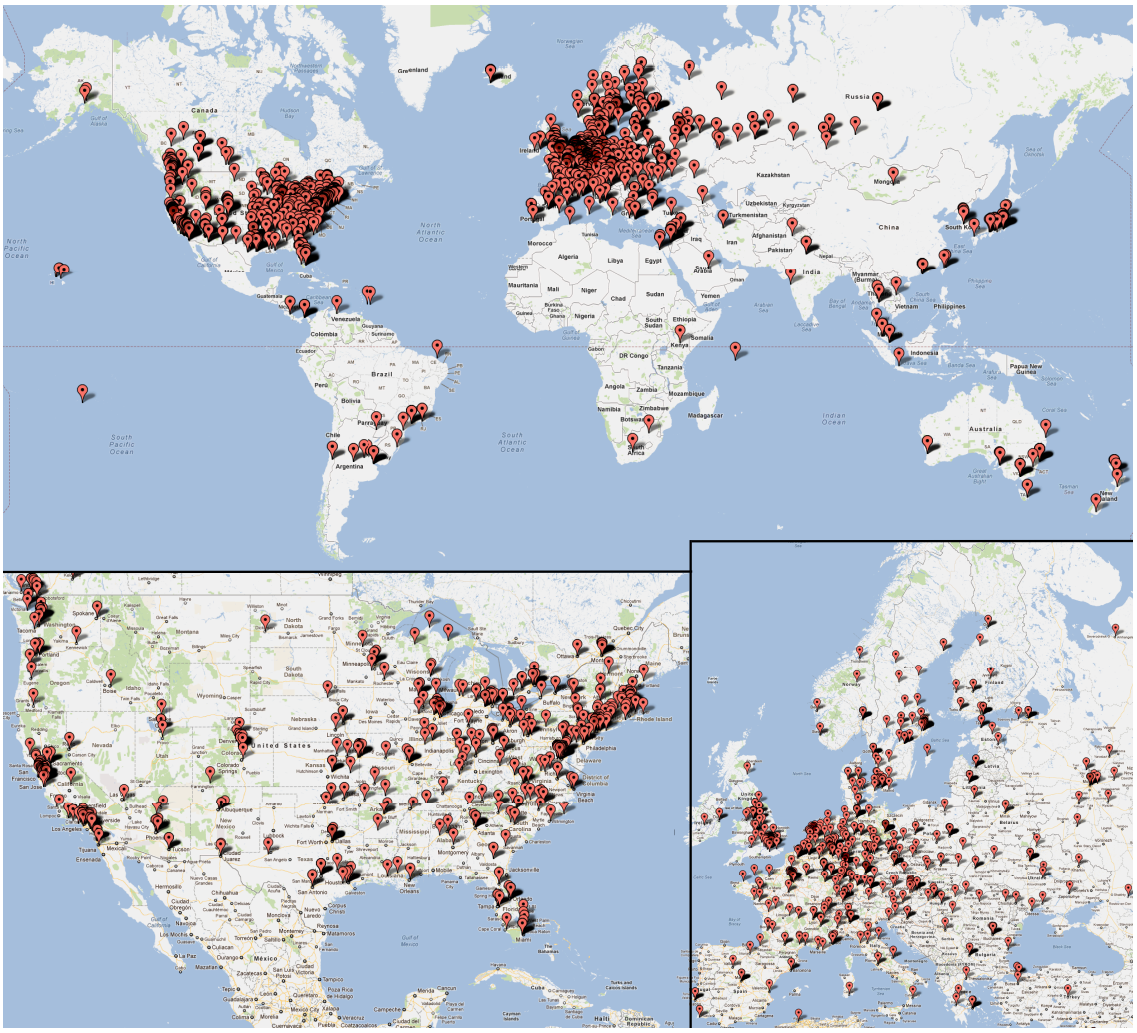

Figure 2: Geographic distribution of Tor relays plotted using May 2012 Tor directory and the Quova GeoIP database.

Additionally, geographic diversity in COR could be comparable to geographic diversity in Tor if the right cloud datacenters were chosen. We examined a copy of the Tor directory downloaded in May 2012. It contained 2772 total entries, 843 of which resided in the United States and 1539 of which resided in Europe. We performed this analysis using the Quova GeoIP database [21], and we've plotted these results in Figure 2. Our analysis shows that the majority of Tor relays reside in the US and Western Europe – the exact same regions where the majority of cloud datacenters reside. It would not be difficult to build COR circuits between these jurisdictions in order to provide jurisdictional arbitrage similar to that provided by Tor. In fairness, there is a significant minority of Tor relays which reside outside of both the US and Europe (112 in Russia, and 178 elsewhere). However, given the randomness with which Tor circuits are chosen, there is still a high probability that all the relays within a circuit would be located in traditional "western" countries.

2.4 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 [28]. 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 [22]. Admittedly, a determined adversary might be able to dynamically track IP addresses within the

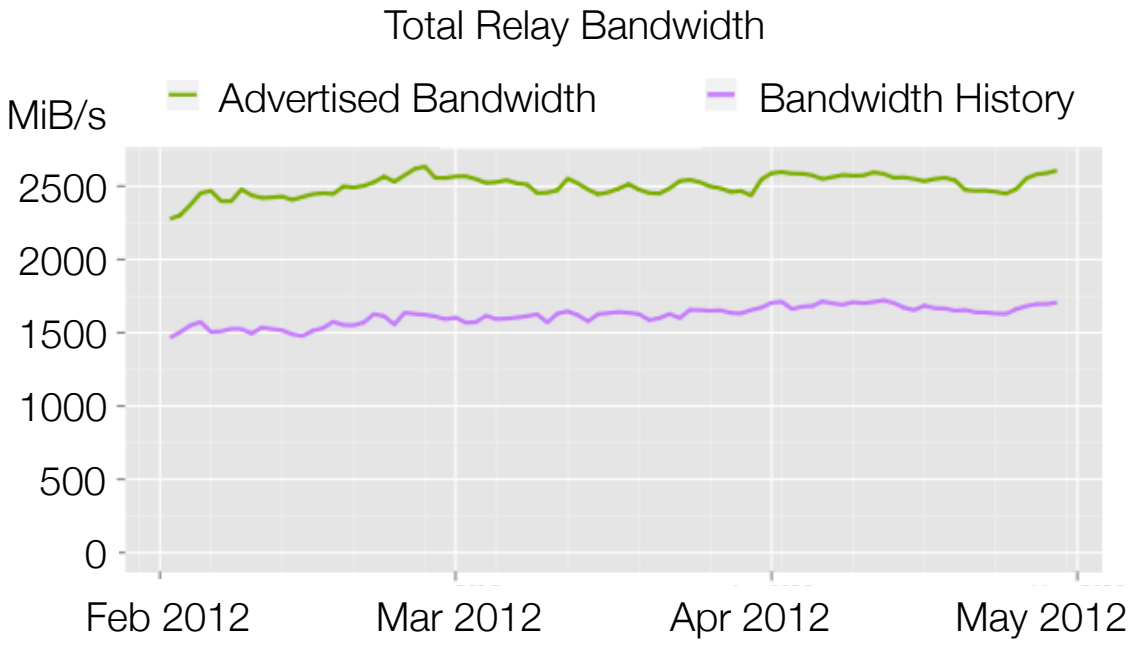

Figure 3: Graph from https://metrics.torproject.org showing aggregate bandwidth usage in the Tor network from February through May 2012.

cloud and "whitelist" certain services. However, this would at best 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.

2.5 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 [24], we estimate that an upper bound for the daily-averaged bandwidth consumed by Tor relays during the spring of 2012 was approximately 1600 MB/s (shown in Figure 3). 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 266 MB/s, or approximately 667 TB/month of end-user traffic. 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 May 2012, 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 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 266 MB/s of traffic, so we can estimate the approximate monthly cost of COR as \$76,900[2] for its 667 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 \$27,000[3]. 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 a new model for circuit construction which is policy aware. We then 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 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

[2] (10,000 GB × \$0.12 × 2) +(40,000 GB × \$0.09 × 2) + (100,000 GB × \$0.07 × 2) + (533,000 GB × \$0.05 × 2) = 683,000 GB at a cost of \$76,900

[3] 667 TB × \$100 / 10TB × 4 = \$26,680

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 [28], 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.

3.2 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.[4]

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

[4]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 [12].

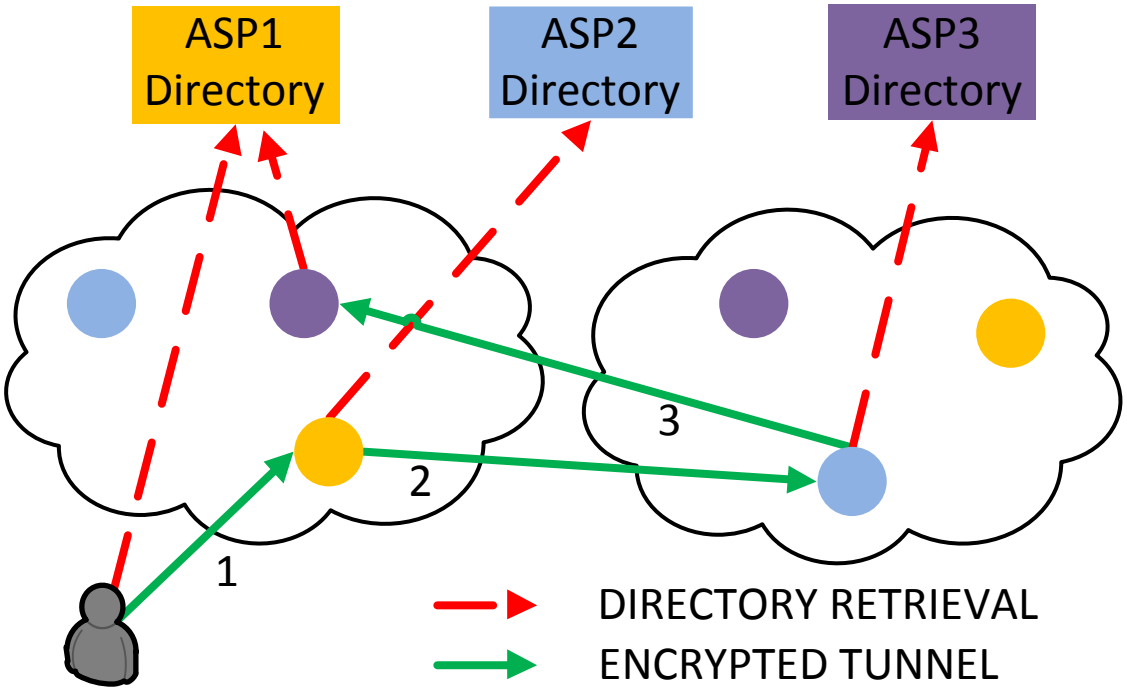

Figure 4: Directory retrieval through the COR bootstrapping network. Relay colors denote the operating ASP.

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. If the user chooses to, he can re-download the directory from the first ASP and ensure that it matches the initial directory he was given.

3.3 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 [8], 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.

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.

3.4 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 Implementation

In this section, we describe our implementation of the prototype COR system. In building our prototype, we attempted to avoid making serious modifications to Tor whenever possible. We chose to do this for a number of reasons, one of which was our desire to retain forward compatibility with new versions of Tor. Additionally, we wanted to avoid re-engineering Tor protocols. We know that there has been significant engineering effort put into the design of Tor's architecture, and making large changes increases the risk of introducing new vulnerabilities.

In order to achieve our goal, we made use of a Tor API called the Tor Control Protocol. Although not an API in the traditional sense, the Tor Control Protocol allows clients to connect to a running Tor process via Telnet and both query data from it and send it commands. Much of the logic for our COR prototype is implemented separately from Tor itself, and communicates with Tor via this protocol.

Our implementation consists of four components – a COR client which runs on an end user's PC, a COR daemon which runs on a COR relay, an e-cash server, and a modified version of Tor.

4.1 Connection Overview

A connection to a COR relay begins by retrieving the directory from the local Tor process via the Tor control protocol.[5] The COR relay then chooses a circuit according to the algorithm described in Section 3.1. Once the COR client has chosen the circuit's path, the client sends an e-cash token to the remote relay's COR daemon. The daemon receives the token, and immediately deposits it at its corresponding e-cash server. The e-cash server unblinds the token, and ensures that the token has not been previously spent. If the deposit is successful, the e-cash server sends a receipt back to the COR daemon to confirm the deposit. Once the COR daemon confirms the deposit was successful, it sends a SHA256 hash of the e-cash token (henceforth referred to as the token

[5]Our current prototype does not implement the bootstrapping protocol. However, this step would not change if that protocol was used.

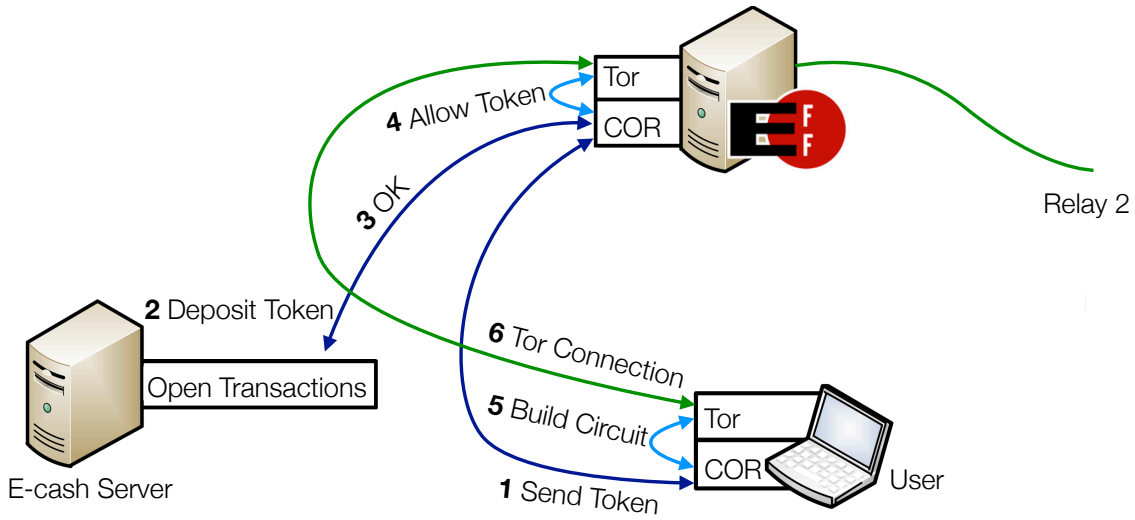

Figure 5: An example of adding a relay to COR with e-cash transactions.

hash) to the Tor process running on the relay via the Tor control protocol. The relay maintains a list of all tokens which it has received, and removes tokens as they are redeemed. At this point, the COR daemon sends a confirmation message back to the COR client telling it that it can now begin building a Tor circuit. When the COR client receives the message, it sends the token hash to the Tor service while instructing Tor to extend the circuit. The Tor service establishes a new circuit with the remote relay, and it embeds the token hash in the connection. When the remote Tor relay receives the incoming connection, it checks to see if the token hash is in its list of allowed token hashes, and either accepts or rejects the connection. Each token can be used only once, so once the Tor relay accepts the connection, it removes the token hash from its list of allowed token hashes.

4.2 COR Client

As described above, the COR client is a small program which runs on an end-user's workstation and works in conjunction with Tor. The COR client is responsible for several actions; when Tor starts, the COR client contacts it using the Tor control protocol and instructs it to stop building circuits using Tor's native circuit selection algorithm. COR then asks Tor for a copy of the Tor directory, and COR then chooses a circuit according to the algorithm described in Section 3.1.

The COR client also handles E-cash operations, which it does using the Open Transactions API. The COR client is responsible for withdrawing an e-cash token from the local "purse" file. All tokens are stored in an encrypted format on an end user's local machine, and the COR client decrypts the token using the user's private key and re-encrypts the token with the recipient's public key.

When the COR client receives confirmation from the remote COR daemon that the token was accepted, the COR client calculates and passes the token hash to the Tor process and instructs Tor to extend the circuit to that relay.

Finally, the COR client also supports the SOCKS protocol for connecting to relays beyond the first hop. A user could be deanonymized if these connections were sent over the regular Internet, so they must be proxied through the COR circuit.

4.3 COR Daemon

The COR daemon is a small server which runs on the same host as the Tor relay and accepts connections from the COR client. When the COR daemon receives an e-cash token, it immediately deposits it at the e-cash server. If the e-cash server accepts the deposit, the e-cash server sends a receipt to the COR daemon. The COR daemon then sends the token hash to the local Tor relay, and it also sends a confirmation message to the COR client telling it that it can continue to build the circuit.

4.4 E-cash Server

We use an open source e-cash server called Open Transactions [20] to handle the implementation of Chaumian e-cash. Open Transactions provides an e-cash server and an API which may be used by third party programs to handle e-cash transactions. This API is used by both the COR client and the COR daemon for e-cash withdrawals and deposits.

Open Transactions uses the Lucre E-cash library [17] to provide support for blinded cryptographic tokens. The Lucre E-cash library is based upon David Wagner's Diffie-Hellman variant of Chaumian blinding. It is believed to not infringe upon David Chaum's patents.

4.5 Modified Tor Client & Server

In order to implement our design of the COR system, it was necessary to make several modifications to Tor. Specifically, we modified the directory server and the control protocol, and we added new features to support tokens and quotas.

In order for COR's circuit construction algorithm to work correctly, it must know the ASP, CHP, and the e-cash public key (called a NYM in Open Transactions) of the relay to which it is connecting. We modified Tor so that each of these parameters can be passed in via the Tor configuration file, and they are then uploaded to the COR directory server along with all other information normally included in the Tor directory. Directory entries in Tor are signed with the relay's private key before being uploaded, so there is no easy way for an attacker to spoof another relay's directory entries.

We made two major modification to the Tor control protocol; the first thing we added was the ability to pass a token hash to Tor when building a circuit. Previously, it was possible to specify an entire circuit of nodes to connect to via the Tor control protocol. However, with our implementation, each node must be added to the circuit sequentially, because it is important to ensure that the relay was added successfully before continuing to add the next node. Unlike in Tor, the position of a relay within a circuit is important in COR, so if adding a relay to a COR circuit fails, it is necessary to rebuild the entire circuit. We also added the ability for a COR daemon to pass tokens to a Tor relay so that the Tor relay will know which tokens are valid when it receives incoming connections. Additionally, we also modified the Pytorctl library, a Python library which provides support for the Tor control protocol, to support these new commands.

Finally, we added quota support to COR. Our implementation uses Tor's "cell" construct as the metric for measuring data, and each Tor cell is 512 bytes. Although the current version implements usage based metering, it would be a simple matter to implement time based metering if that was the desired metric. Unfortunately, our current implementation does not support increasing the quota on an already open circuit. In that case, the client must establish a new circuit in order to continue accessing COR. It would be possible to add the ability to increase an existing circuit's quota, but we leave that for future work. Alternatively, a COR user can just build a new circuit whenever his current circuit is about to expire.

Circuit
R1 → R2 → US1 → US2
US1 → US2 → R1 → R2
R1 → R2 → EU1 → EU2
EU1 → EU2 → R1 → R2
US1 → US2 → R1 → R2 → EU1 → EU2

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

5 Evaluation

Tor performance is one of the major factors inspiring our work on COR. Thus, our 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, as well as the overhead introduced by COR's new circuit construction process.

5.1 Individual File Downloads

To evaluate the performance of COR when downloading individual files, we use the Tor-supplied TorPerf [24] 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 6. 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.

5.2 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

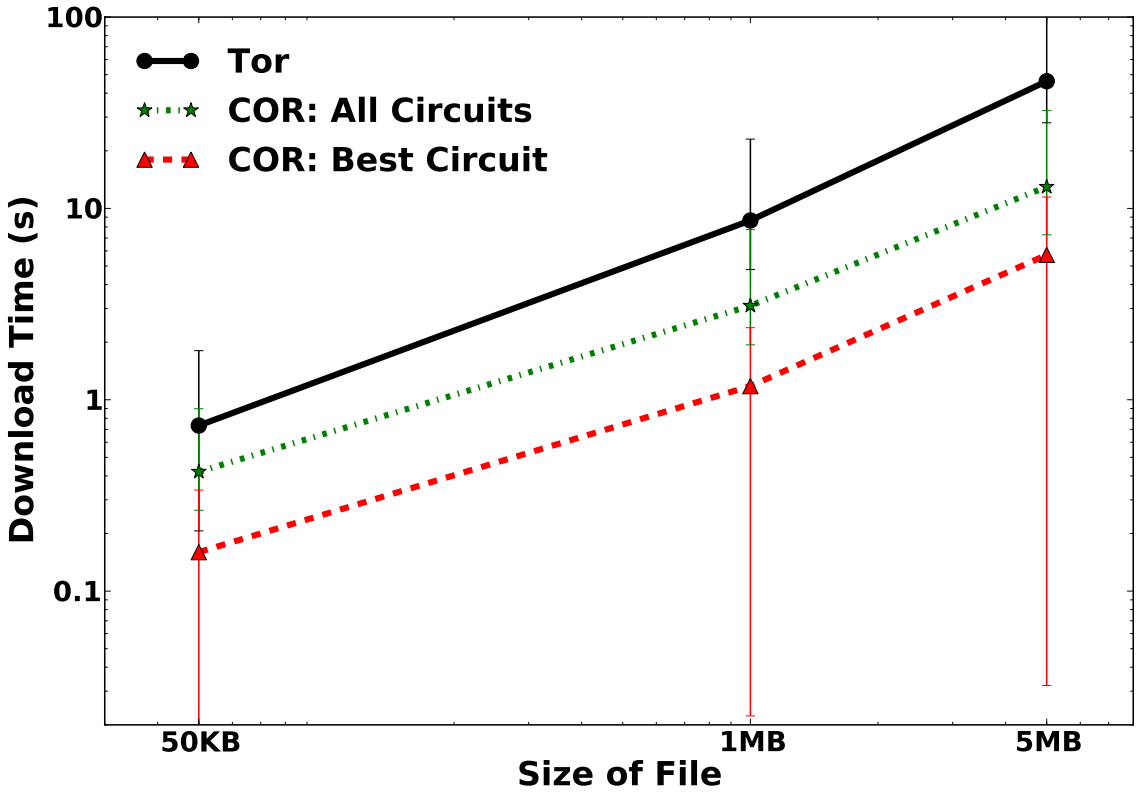

Figure 6: Median download time, measured between the first and last byte received of an HTTP response. Error bars illustrate the quartiles.

the top 10 Alexa.com domains,[6] 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 7 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.[7]

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

[7]It is noteworthy that the Chinese web site www.baidu.com was faster over COR than over a direct connection; we surmise the link from the Amazon and Rackspace datacenters to China are better than the direct link.

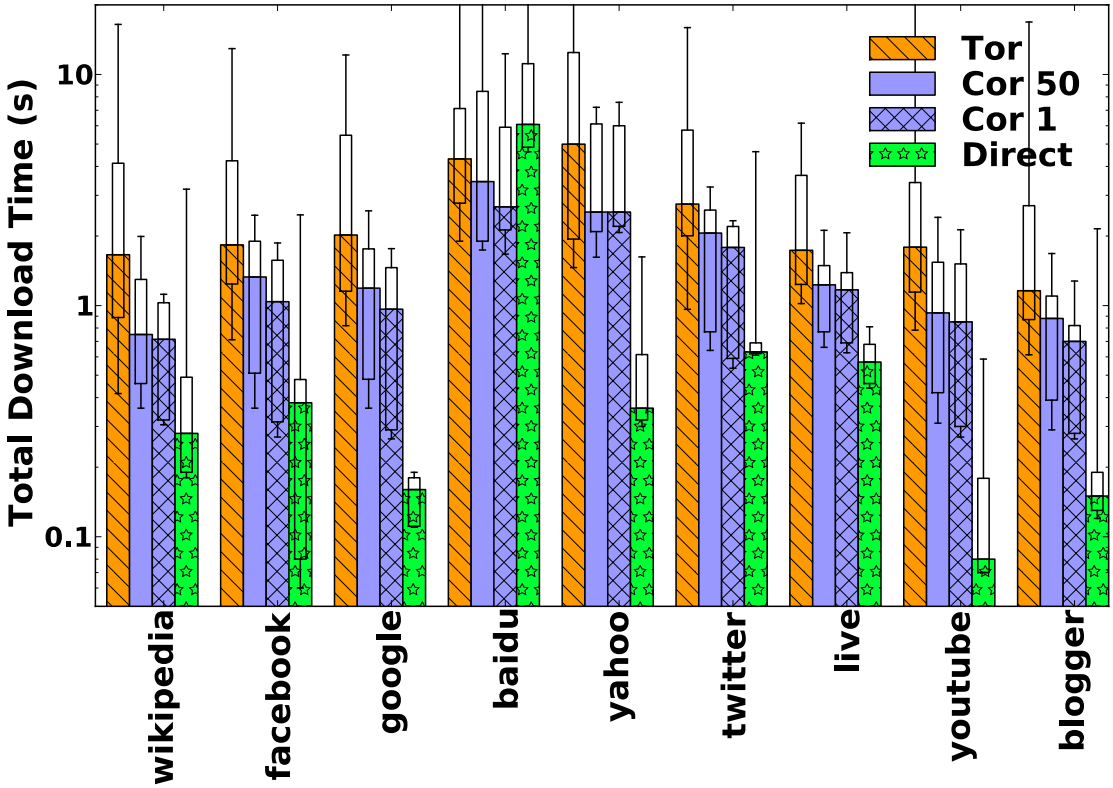

Figure 7: Median download times for a selection of popular websites. Error bars show 90th and 95th percentile download times.

5.3 Concurrent Users

Next, 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 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 [2] and evaluating the bandwidth of the transfer. The results are shown in Figure 8. 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+

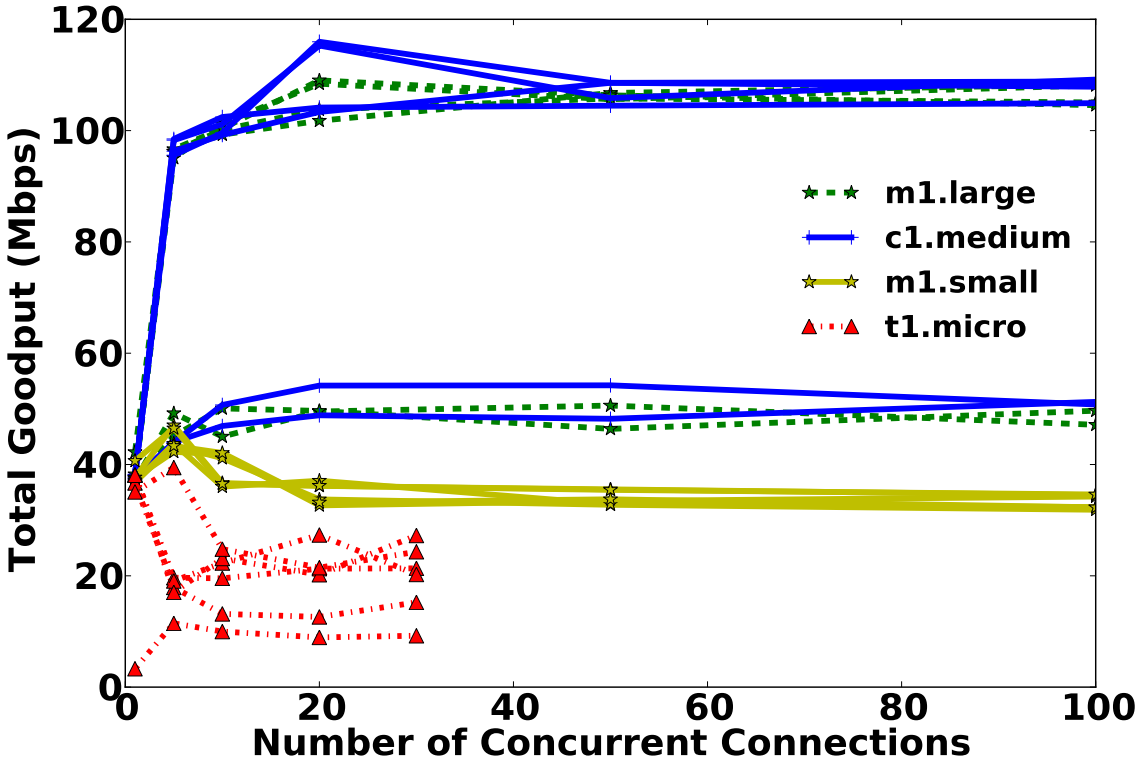

Figure 8: Aggregate COR bandwith through various types of Amazon EC2 instances, to establish the number of concurrent users that a single cloud-hosted relay can support.

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.

5.4 Circuit Construction

Our last experiment evaluated the overhead introduced to circuit construction by adding e-cash transactions and authentication. We performed two versions of this experiment in order to evaluate

Action	One Hop Circuit	Two Hop Circuit
Start	0.0	0.0
Read token from purse file	0.03746	0.01996
Decrypt and re-encrypt token	1.72509	2.47453
Print status message	1.77112	2.50021
Caculate SHA256 hash	1.77118	2.50029
Open regular/SOCKS socket	1.77120	2.50031
Recieve relay reply	2.56128	3.53595
Open circuit command complete	2.56593	3.53823

Table 2: Timing benchmarks for COR circuit construction as measured at the COR client. Time is shown in seconds.

Action	One Hop Circuit	Two Hop Circuit
Recieve connection	0.0	0.0
Push token to purse object	0.03323	0.07464
Deposit purse at server	0.78445	1.02625
Send add token message to Tor process	0.78935	1.03125
Send success message to COR client	0.78935	1.03125

Table 3: Timing benchmarks for COR circuit construction as measured at the COR daemon. Time is shown in seconds.

circuit construction overhead. In the first experiment, we added a single relay to a COR circuit. Since this was the first relay in our path, this connection was not proxied through COR. In our second experiment, we again added a single relay, but we added the relay to an existing one hop circuit. We did this so that we could evaluate the overhead introduced by the SOCKS proxy. In both cases, the experiments were run on a local area network in order to minimize network latency. Our results are shown in Tables 2 and 3.

One of the most expensive operations that the client performs involves changing the ownership of the e-cash token. All e-cash tokens are stored encrypted in the user's wallet file, and they must be decrypted and re-encrypted with the recipient's public key before being sent over the network. The other actions introduce minimal overhead, until the token is sent over the network.

Once the remote COR daemon receives the token, its most expensive operation is sending it to the e-cash server. This takes almost a second, even on a local area network. There is some overhead from decrypting the token (shown in the table as storing token in purse), but it is on the order of a few milliseconds.

Finally, there may be other load factors unaccounted for in this experiment. This experiment

was performed on a high performance cluster, but it was a shared resource, and it is possible that other users/processes could have induced CPU load.

6 Alternative Models

In this paper, we've proposed one model for coupling cryptographic anonymous payments with cloud infrastructure in order to provide high-speed anonymous web browsing. However, this model is not the only model possible, and in this section we describe other possible designs using the same system components.

6.1 QOS Based Onion Routing

Current implementations of Tor treat all traffic similarly – giving neither higher or lower performance to different circuits. However, it would be feasible to modify Tor so that it had multiple queues for incoming traffic, and where certain kinds of traffic recieved higher priority. In this model, the high priority traffic could orginate with user's who have paid for access (like in COR) or from some other mechanism. One advantage of this model is that participating relays could contribute unused bandwidth they have to the regular Tor network, while still delivering high-speed anonymous browsing to their clients.

6.2 Private Onion Routing

COR proposes coupling anonymous access control with cloud infrastructure, but it would also be possible to build a traditional Tor network that used access control. In this model, end users would run Tor relays, but like in COR, they would accept tokens from incoming connections. The users would then be able to spend the tokens themselves for anonymous browsing, or to resell them at market value. This scheme has several advantages, because it retains the network and geographic diversity currently present in Tor, while removing users who do not contribute bandwidth to the Tor system. Unfortunately, this model has downsides in that many users of Tor might not be able to host a Tor relay, especially those users who are in countries with censorship or network level monitoring.

6.3 Anonymous VPNs

A third possible model for coupling access control with anonymous browsing would be to couple e-cash payments with traditional anonymous VPN services. There are several existing VPN services which aim to provide anonymity by keeping no logs of the traffic which their customers send. These VPN providers could accept e-cash tokens in lieu of their traditional authentication mechanisms, and then build circuits across multiple legal jurisdicitions. This would provide significant jurisdicitional arbitrage to the VPN's users, with little change to the VPN's current usage model. Of course, in this scenario, the VPN provider would still be able to de-anonymize any of its users, but this is not different from the VPN providers' current threat model.

7 Related Work

COR couples anonymous payments with the use of cloud infrastructure to provide high speed anonymous browsing. Anonymous payments, anonymous browsing, and cloud computing all have rich research literatures, but no existing work couples these research areas together as COR does.

The concept of untraceable electronic currency was first introduced by Chaum [7], and has since been studied significantly. Multiple schemes for anonymous payments have been proposed, including a scheme promoted by Chaum's company DigiCash. COR does not propose a new design for the anonymous payment infrastructure, and multiple systems could be used with COR as long as they provide both blinded currency and protection from double spending.

COR diverges more significantly from existing systems for Internet anonymity. Multiple systems have been proposed over the past two decades for providing anonymity in the network, the first of which [11, 15, 18] focused on e-mail. These systems ensure anonymity, but are inadequate for high bandwidth or latency sensitive applications.

To address this problem, VPNs like Anonymizer [4] were proposed. Two characteristics make these kinds of solutions unattractive: (i) users have to *trust* the proxy, and (ii) a capable attacker can still demultiplex data if he is able to eavesdrop on all incoming and outgoing traffic. To counter this, later designs [6, 10, 23] introduce the use of multiple proxies.

The work upon which COR is based, onion routing, was first introduced in [14]. The Tor system was introduced in the follow on paper [13], and we use much of its codebase for the COR

implementation.

More recent and more closely related work has suggested the use of cloud VMs as Tor relays, specifically, Dust Clouds [19]. However, unlike COR, Dust Clouds does not support the creation of an economic ecosystem that would allow users to purchase cloud-based anonymity services securely.

There have been recent proposals for using anonymous payments with network services. Proposals like XPay [9] and Par [3] offer more complex solutions than those needed by COR. XPay deals with micropayments and does not examine the challenges relating to a cloud based deployment. Par assumes a single central bank which is not a model that applies to COR. Bitcoin is not appropriate for use as a token because all transactions are logged and stored within the Bitcoin network [5]. However, Bitcoin could be used as a currency for purchasing tokens, much like any other currency that an ASP chooses to accept.

8 Future Work

Our current COR implementation fully implements the basic COR design, but we have not yet implemented the bootstrapping network or any of the alternative models which we described in Section 6. Our current implementation is suitable as a research prototype, but would need more testing before being deployed with real world users.

Ultimately, we would like to see a COR system deployed at scale. To do this, we would need several ASPs to participate in the system. We would also need to find CHPs who would willingly host COR traffic, although this appears to be an easy problem to solve. (There are numerous Tor relays which are currently deployed on Amazon EC2.)

Finally, bootstrapping COR and similar networks in censoring environments is still an open research problem. Telex [25] and related systems offer one possible approach to this problem, but they require performing deep packet inspection at line rate at major peering points in the network, which presents a significant deployment challenge.

9 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. Our 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] Amazon S3. `http://aws.amazon.com/s3/`, 2011.

[3] E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. M. Bellovin. Par: Payment for anonymous routing. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, PETS '08, 2008.

[4] Anonymizer. `http://www.anonymizer.com/`.

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

[6] P. Boucher, A. Shostack, and I. Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), February 1981.

[8] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, 1982.

[9] Y. Chen, R. Sion, and B. Carbunar. Xpay: practical anonymous payments for tor routing and other networked services. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, WPES '09, 2009.

[10] W. Dai. Pipenet 1.1. *Usenet post, August*, 1996.

[11] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

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

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[14] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2), Feb. 1999.

[15] C. Gülcü and G. Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, February 1996.

[16] HotSpotVPN. `http://www.hotspotvpn.com/`.

[17] Lucre E-Cash Library. `http://anoncvs.aldigital.co.uk/lucre/`.

[18] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.

[19] R. Mortier, A. Madhavapeddy, T. Hong, D. Murray, and M. Schwarzkopf. Using dust clouds to enhance anonymous communication. In *Eighteenth International Workshop on Security Protocols*, 2010.

[20] Open Transactons. `https://github.com/FellowTraveler/Open-Transactions`.

[21] Quova GeoIP Database. `http://www.quova.com/`.

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

[23] M. Reiter and A. Rubin. Anonymous web transactions with crowds. *Communications of the ACM*, 42(2), 1999.

[24] Tor Metrics. `https://metrics.torproject.org/`.

[25] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, August 2011.

[26] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *In IEEE Symposium on Security and Privacy*, 2004.

[27] X. Yang, D. Wetherall, and T. Anderson. Tva: a dos-limiting network architecture. *IEEE/ACM Transactions on Networking*, 16(6), Dec. 2008.

[28] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, 2010.