Provably Avoiding Geographic Regions for Tor's Onion Services

Arushi Arora[1], Raj Karra[1], Dave Levin[2], and Christina Garman[1]

[1] Purdue University
{arora105, karra0, clg}@purdue.edu
[2] University of Maryland
dml@cs.umd.edu

Abstract. Tor, a peer-to-peer anonymous communication system, is one of the most effective tools in providing free and open communication online. Many of the attacks on Tor's anonymity occur when an adversary can intercept a user's traffic; it is thus useful to limit how much of a user's traffic can enter potentially adversarial networks. Recent work has demonstrated that careful circuit creation can allow users to provably avoid geographic regions that a user expects to be adversarial. These prior systems leverage the fact that a user has complete control over the circuits they create. Unfortunately, that work does not apply to onion services (formerly known as "hidden services"), in which no one entity knows the full circuit between user and hidden service.
In this work, we present the design, implementation, and evaluation of `DeTor`$_{OS}$, the first provable geographic avoidance system for onion services. We demonstrate how recent work to build and deploy programmable middleboxes onto the Tor network allows us to take existing techniques like these and deploy them in scenarios that were not possible before. `DeTor`$_{OS}$ is immediately deployable as it is built using programmable middleboxes, meaning it does not require either the Tor protocol or its source code to be modified.
This work also raises a number of interesting questions about extensions of provable geographical routing to other scenarios and threat models, as well as reinforces how the notion of programmable middleboxes can allow for the deployment of both existing and new techniques in novel ways in anonymity networks.

Keywords: Programmable anonymity networks · Tor · Onion Services · Privacy.

1 Introduction

The ability to achieve freedom of speech anonymously and access resources privately has now become an important part of our society. Tor, one of the most popular and widely used anonymous communication networks today, is used by people across the globe who wish to share or access systems without revealing their identity. In addition, Tor's *onion services* let users host content anonymously (i.e. without disclosing the host server's IP). This is critically important not just in the face of internet censorship and hosting of services like anonymous dropboxes for whistleblower submissions, but also for regular users who might want to protect their privacy online.

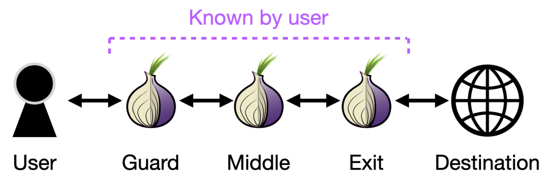
Tor is designed under the realistic assumption that no adversary has a *global* view of the network [5]. However, even under this relaxed threat model, there are still very powerful *routing-capable* nation-state adversaries that can manipulate, inspect, and correlate traffic crossing their borders. In other words, these attackers can censor Tor traffic and launch powerful deanonymization attacks against Tor users (including onion service hosts). Some of the first known attacks on onion services include timing analysis, service location attacks, and distance attacks, which expose the location of a server hosting an onion service [12, 20]. Further, circuit fingerprinting attacks [14] attempt to recognize the circuits involved in communicating with an onion service and then perform a website fingerprinting attack [27] on the identified circuits to deanonymize the target service with high accuracy.

While, for Tor clients, some work has sought to deal with such attackers by making traffic appear innocuous to them, others have proposed avoiding these attackers altogether [13, 15, 17]. Although the Tor protocol provides a way to allow users to specify certain countries to avoid, this avoidance is not certain [21]. For instance, one study found that circuits excluding US Tor nodes only bypassed the US 12% of the time [17].

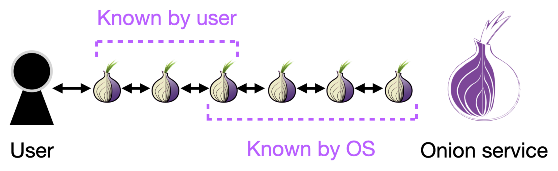
These efforts have led to techniques that allow users to specify *forbidden* geographic regions, and to construct circuits that *provably* bypass these regions. After a round-trip of communication, the idea is to return a proof verifying that packets could not have traversed the forbidden region. This proof combines proof of some of the places where the packet *did* go, combined with the fact that information cannot travel faster than the speed of light as an "alibi," thereby showing where the packet *could not* have gone.

Unfortunately, no prior work has managed to extend these provable avoidance techniques to onion services. In general, extending such architectures to onion services seems to inherently be a hard problem given the design of the interaction between clients and an onion service. As shown in Figure 1, there is a 6-hop circuit between a client and an onion service, of which both the parties know only their side of the respective 3-hop circuit. Since both parties involved are unaware of the other side's respective circuit to ensure anonymity, it is a challenge for either side to gain assurances of the other half without a loss of anonymity.

Our proposed approach, which we call $\texttt{DeTor}_{OS}$, allows an onion service host as well as their respective users to verify whether their traffic successfully evaded certain regions. We are able to extend this functionality to onion services by leveraging recent advances that introduce programmability to Tor (and other similar anonymity systems), which allows a user to upload and execute code on willing Tor relays [22]. This paradigm allows us to do something that was not possible before: build $\texttt{DeTor}_{OS}$ as a (trusted) function which can compute on data from both parties in a confidential manner. Our design also has the benefit of being immediately deployable. Because $\texttt{DeTor}_{OS}$ leverages programmable



(a) Traditional Tor circuits



(b) Onion service Tor circuits

Fig. 1. Whereas a user knows every hop in the circuit they create, circuits to onion services are created collaboratively, and thus no one entity knows the full circuit. This makes provable avoidance difficult to achieve.

anonymity networks, no changes are required to the underlying Tor source code or protocol. We also show that, under reasonable assumptions, it protects the anonymity of both the onion services and their users.

Contributions. Our contributions in this work are as follows. We aim to improve the usability of Tor's onion services for its host as well as its respective users. We achieve this by providing the first realization for provable geographic avoidance for onion services that is immediately deployable and requires no changes to the underlying Tor code. We present the design, implementation, and evaluation of $\texttt{DeTor}_{OS}$, a set of techniques that aim to provably guarantee avoidance for onion services. While we leverage existing techniques to provide provable avoidance, we are the first to demonstrate how this can be done for onion services.

Roadmap. The organization of this paper is as follows. In Section 2, we provide a brief discussion on related work and background knowledge of the core ideas in the paper, as well as discuss the threat model for our architecture. Next, we discuss the design of $\texttt{DeTor}_{OS}$ in Section 3 and its security analysis in Section 4. We evaluate our work in Section 5. We discuss the implications of our design and pave a path for future directions in Section 6, discuss ethical considerations in Section 7, and conclude in Section 8.

2 Background and Related Work

Tor. Tor is a peer-to-peer overlay network based on onion routing that allows its users to browse the internet anonymously. This low-latency TCP-based communication service lets a client, who runs an Onion Proxy (OP), build a *circuit* (3-hop by default) consisting of volunteer Tor relays called an Onion Router (OR)- *guard* (connects the source), *middle*, and *exit* (connects the destination) nodes. See Figure 1a. The idea is to then encrypt the client's message (that needs to be sent to the destination) first with the unique symmetric key shared with the *exit* node followed by *middle* and *guard* node. This triple-encrypted ciphertext is then passed to the *guard* node, which decrypts ("peels off") it using the same shared symmetric key. The *middle* and *exit* nodes then "peel off" their respective layers until the message reaches the destination. This technique, therefore, maintains the confidentiality of the message, obscuring it from anyone trying to intercept it between two ORs. The Tor protocol by default prioritizes high-bandwidth relays for circuit construction and selects them from different subnets. The client constructs these circuits preemptively and is aware of all the chosen relays, whereas the ORs are only aware of their immediate successor or predecessor in the circuit. The OP is responsible for multiplexing TCP streams across circuits. Many such streams can share the same circuit. Tor protocol chooses ORs (almost) uniformly at random to construct its circuits.

Onion Services. An onion service allows its host to share information across the internet while maintaining its anonymity i.e. without revealing the identity or location of the host server. To establish an onion service, its host *Alice* first generates a public key pair and selects Tor relays to be its introduction points (IPs). *Alice* then publicizes her service (signing it with her private key) and forms a circuit to her IPs. *Bob*, who wants to visit *Alice*'s service, does so via Tor. To establish a connection to the onion service, first, he would select an onion router (OR), which is a Tor relay, to be his rendezvous point (RP). *Bob* then sends a cookie to the RP and builds a circuit to *Alice*'s IP sending it a message encrypted with *Alice*'s public key and starting a DH (Diffie Hellman) handshake. This message, which contains *Bob*'s cookie, and information about himself and RP, is forwarded to *Alice* who can then connect with *Bob* anonymously by building a circuit to RP. In this case, *Alice* would send the second half of the DH handshake, cookie, and hash of the session key. This establishes the anonymous stream between *Alice* and *Bob*. See Figure 1b.

Bento. `Bento` [22] introduces programmability to anonymity networks (Tor as of now) by allowing users to execute tiny code snippets (called *functions*) on Tor relays thus improving a user's anonymity and performance and Tor's usability. This architecture runs on top of Tor and is immediately deployable. These in-network middleboxes can support numerous jobs like load-balancing, sending cover traffic, sharding files, and browsing the web with just a few lines of code. `Bento` also introduces a middlebox node policy that specifies resources and tasks that a `Bento` server can provide to its users. This ensures that Tor relays enacting as `Bento` servers are protected from the functions they run. Similarly, this system considers that some Tor relays support trusted execution environments (TEEs) which therefore prevent a third-party `Bento` server to introspect on any user function or its relevant data, as well as enforce correctness of execution for the function [8].

Provable Geographical Avoidance. Powerful nation-state adversaries can influence Tor routing into and out of their borders [26]. This allows them to censor and deanonymize traffic through correlation attacks [10,16]. Li et al. [17] introduced DeTor, a technique for constructing Tor circuits that provably avoids geographic regions of the user's choosing, based on Alibi routing [15]. Kohl et al. [13] and Ryan et al. [25] overcame some of DeTor's limitations by extending to asymmetric paths and providing more accurate node locations, which was later further refined by Ryan et al. [25]. At a high level, all of these make use of the same basic proof structure: clients prove where the traffic *did* go (based on the locations of the relays on the circuit), and combine that with latency information to infer where the traffic *could not have* gone (based on the fact that information cannot travel faster than the speed of light).

These proofs are calculated as

$$\begin{aligned} &\min_{f\in F}[R(s,f)+R(f,a)]+R(a,t)\geq \\ \frac{3}{2c}\cdot &\left(\min_{f\in F}[2\cdot D(s,f)+2\cdot D(f,a)]+2\cdot D(a,t)\right) \end{aligned} \tag{1}$$

where $D(x,y)$ is the great circle distance between hosts located at x and y; $R(x,y)$ denotes the RTT between x and y; F represents a forbidden region and $f\in F$ refers to a forbidden geographic coordinate; a is a relay that is not in F; s and t represent a source and destination node respectively.

DeTor provides proof for two modes of avoidance [17]. *Never-once*, which desires to fight fingerprinting [14] and censorship attacks [19] by verifying that a packet, passing through a Tor circuit, never transited a particular geographic region, even once. And *Never-twice*, which aims to resist deanonymization attacks [3, 9, 10, 18], wherein an adversary needs to witness a packet twice: at the entry leg (from client to entry node) and the exit leg (from exit to destination). This technique confirms that an adversary does not appear on two non-contiguous portions of the Tor circuit.

2.1 Threat Model

Our network-level threat model is similar to Tor's. We consider our attacker to be a powerful nation-state adversary. Such a routing-capable adversary is unable to have a global view of the Tor traffic, but can observe, control and censor traffic in their respective local geographic regions. They may be able to achieve this by advertising themselves as Tor relays. In addition, this adversary is not restricted to a specific region or nation and may be able to collude with other (non-neighboring) countries.

`Bento` involves executing *functions* on a third-party machine (the `Bento` server). Following the original `Bento` design, we assume that the host node itself could be malicious, meaning that it can both try to tamper with or gain information about the avoidance computation and its inputs and outputs, as well as try to manipulate the inputs to alter the results. We assume that some of these servers will have secure TEEs, such as Intel SGX[3], prohibiting the host machine to access the executing *function* and its relevant data, as well as ensuring correctness of execution. As thus far all known TEE vulnerabilities have been patched by their respective vendors, we therefore assume that these TEEs are not fundamentally flawed, and that such an environment can indeed provide a secure enclave and is safe for running code, denying a malicious attacker/host the ability to introspect the executing *function*. We discuss the implications of the use of TEEs further in Section 6, but note that in this work we do not rely on any additional assumptions from the TEE, and inherit (and can make use of) `Bento`'s support for remote attestation [1, 11], which allows a client to verify that the `Bento` server is truly running inside an enclave and that the current TCB version as been patched against all known vulnerabilities.

The $\texttt{DeTor}_{OS}$ architecture itself employs an honest-but-curious model for the client and server. We therefore assume that both the client and the onion service are faithful to the $\texttt{DeTor}_{OS}$ protocol even though they attempt to learn what information they can.

3 $\texttt{DeTor}_{OS}$ Design

$\texttt{DeTor}_{OS}$ extends the DeTor proofs and computations [17] to onion services. This work introduced the idea of never-once proofs, which involved calculating $D_{\mathsf{min}}(x_1, \ldots, x_n)$: the shortest possible great-circle (geographic) distance along a circuit $x_1 \rightarrow \cdots \rightarrow x_n$, and converting this into the shortest possible traversal time by dividing it by $2c/3$ (the fastest speed at which information travels on the Internet). They also introduced never-twice proofs, which involve computing geographic ellipses denoting where in the world the packets could have traversed over each leg of the circuit, and then determining whether the entry and exit leg ellipses intersect. If they did not intersect, then never-twice avoidance was successful (see [17] for more details on the exact calculations). Performing these computations requires knowing the precise locations of each hop on the circuit. While this is straightforward for traditional Tor circuits as the client chooses these nodes, when working with onion services, the client cannot know the entire combined circuit (namely the onion service's half of the circuit is hidden from them), making this a hard problem. This section presents the solution for this problem, that is, the design for the $\texttt{DeTor}_{OS}$ never-once and never-twice functions.

3.1 $\texttt{DeTor}_{OS}$ Overview

The central idea behind $\texttt{DeTor}_{OS}$ is to use a semi-trusted `Bento` function that sits between the client and onion service to which both can upload their half-knowledge of the circuit, and that can then perform the computations and determine whether the circuit achieved never-once or never-twice avoidance. Critically, although the function reveals whether or not avoidance was achieved, it

[3] We note that, as discussed in [22], the `Bento` architecture is not bound to SGX and can work with any TEE that supports similar functionality [2].

does not reveal either side's inputs (much like secure multiparty computation). The overall design is presented in Figure 2.

We break this design down into two different sub-functions, one for never-once avoidance and one for never-twice avoidance, though the core protocol is the same for both. A client, Alice, first either identifies a `Bento` node that is running the $\texttt{DeTor}_{OS}$ function or else uploads it to a chosen node. Before running the $\texttt{DeTor}_{OS}$ protocol, both Alice and Bob first perform a TLS handshake with the $\texttt{DeTor}_{OS}$ function to establish a secure channel against a malious node operator (see Section 3.4), and, optionally, ask the `Bento` server to attest to the correctness of its code base. This provides them with strong guarantees of the correctness and confidentiality of their subsequent proofs of avoidance. Alice would then run the desired $\texttt{DeTor}_{OS}$ protocol (presented in detail in Sections 3.2 and 3.3) as part of connection establishment with Bob's onion service before communicating with it further[4].

Computation models. We also introduce two different models of computation: the *`Bento`-side computation* model and the *local computation* model. In the `Bento`-side computation model, the client and OS simply upload all necessary circuit information to the $\texttt{DeTor}_{OS}$ function, which then performs the computations and returns the result. In the local computation model, the client and OS perform the bulk of the avoidance proof locally and then upload only their results to the $\texttt{DeTor}_{OS}$ function, which then computes the final result. We discuss each of these models further in the respective never-once and never-twice sections. These different models trade off trust in the $\texttt{DeTor}_{OS}$ function and `Bento` server for increased computation for the client and OS.

3.2 $\texttt{DeTor}_{OS}$ Never-Once function

The main objective of the never-once avoidance technique is to gain assurance that a packet or its response could not have passed a user-specified forbidden region F during a round-trip transmission. The idea is to first obtain the end-to-end round-trip time R_{e2e} of packets traversed through the selected Tor relays (entry (e), middle (m), and exit (x)). We also take into consideration the case where the packets could have gone through the forbidden region, calculating the shortest possible time necessary to go through each circuit and the forbidden region R_{min} as

$$R_{min} = \frac{3}{2c} \cdot \min \begin{cases} 2 \cdot D_{min}(s, F, e, m, x, t) \\ 2 \cdot D_{min}(s, e, F, m, x, t) \\ 2 \cdot D_{min}(s, e, m, F, x, t) \\ 2 \cdot D_{min}(s, e, m, x, F, t) \end{cases} \tag{2}$$

where δ acts as an extra buffer against irregular delays. We then check if

$$(1 + \delta) \cdot R_{e2e} < R_{min} \tag{3}$$

[4] Assuming Bob's OS supports the $\texttt{DeTor}_{OS}$ protocol. In our current honest-but-curious model, we can provide no guarantees if the OS refuses to participate.

is satisfied, which, therefore, proves that the packets could not have possibly transmitted through F. Otherwise, one cannot decipher if the packets traversed the F or simply suffered a delay. We now present our never-once design for onion services and how it incorporates these computations, referencing the steps in Figure 2.

The `DeTor`$_{OS}$ never-once function, when loaded and executed by a client on a `Bento` server, first accepts both client- and onion service-side circuits. In other words, a client Alice, who wishes to communicate with Bob's onion service, would first send her entry and middle nodes and the RP to the `DeTor`$_{OS}$ function that she has uploaded to a `Bento` server, along with her desired forbidden regions. Simultaneously, Bob would also send his part of the circuit, comprising of his entry, middle, and exit nodes to the `DeTor`$_{OS}$ function (*Step 1*). The `DeTor`$_{OS}$ function then performs the aforementioned never-once computations for the forbidden region as specified by the client (*Step 2*). The function then attests to whether Alice's communication with Bob would avoid the forbidden region as specified by her (*Step 3*). Alice would then receive an attestation of avoidance. Alice and Bob communicate normally after a successful attestation (*Step 4*). This realizes our *`Bento`-side computation* model.

Optionally, Alice and Bob may also choose to perform the never-once computations locally (the *local computation* model), and then only upload the result of this in *Step 2*. This trades off trust in the `DeTor`$_{OS}$ function and `Bento` server (as neither party needs to send their circuit information now) for increased computation (as they must now do the calculations on their own).

3.3 `DeTor`$_{OS}$ Never-Twice function

The main objective of the never-twice avoidance technique is to gain assurance that a packet or its response could not have passed a user-specified forbidden country C on both the entry (C_e) and exit legs (C_x) of the Tor circuit. To prove this case, one needs to verify that the entry (focal point s and e and radius $\frac{3}{c} \cdot (R_{e2e} - R_m) - D(x,t)$, where $R_m = \frac{3}{c} \cdot D(e,m,x)$) and exit leg (focal point x and t and radius $\frac{3}{c} \cdot (R_{e2e} - R_m) - D(s,e)$) ellipses do not intersect, thereby, denying the possibility for the same country to have been traversed twice. If these entry and exit leg ellipses intersect then one must additionally verify the following condition to prove that same countries were not traversed twice.

$$\forall F \in C_e \cap C_x : (1+\delta) \cdot R_{e2e} < \frac{3}{c} \cdot (D_{min}(s,F,e) + D(e,m,x) + D_{min}(x,F,t)) \quad (4)$$

Similar to never-once, we accomplish never-twice avoidance for onion services by building it as a function that is uploaded to a `Bento` server. The `DeTor`$_{OS}$ never-twice function first accepts both the entry legs of the client- and onion service-side circuits (*Step 1*). The `DeTor`$_{OS}$ function then computes the set of countries that Alice and Bob's entry legs could have gone through (*Step 2*), and then returns the intersection of the two sets (*Step 3*). Alice then receives the intersection. If this intersection is empty, Alice and Bob communicate normally

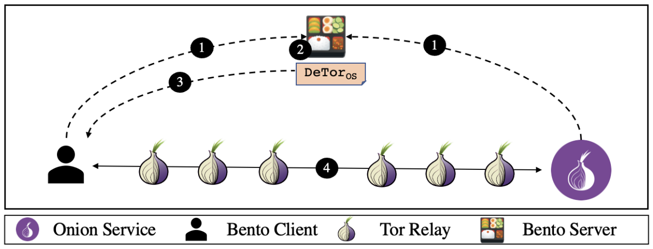


Fig. 2. The `DeTor`$_{OS}$ Protocol for never-once and never-twice avoidance. Both parties send their circuit information to a `Bento` server which is running the `DeTor`$_{OS}$ function. The server then computes the desired avoidance proof and sends it to the client. If successful, the client then begins communicating with the onion service.

(*Step 4*), otherwise Alice can choose to run never-once for the countries in the intersection to gain additional information.

In the case of the *local computation* model, the client and onion service each compute the set of countries that the entry legs of their circuits go through and upload these sets to the function (*Step 2*), which then computes the intersection.

3.4 Ensuring Correctness of Input Data

As we do assume the `Bento` operator (i.e., the host Tor node and other networking infrastructure) itself can be malicious, we must also ensure that the correct data and circuit information is able to reach our `DeTor`$_{OS}$ function, and that a corrupt operator cannot substitute it for their own data and thus corrupt the enclave calculations. We can achieve this by simply having the client and OS both establish a TLS connection with the `DeTor`$_{OS}$ function (running in the protected enclave[5]) prior to passing in the circuit information, thus allowing them to pass all data directly to the enclave and `DeTor`$_{OS}$ through this secure channel. This can be done as part of the initial (pre-computation) setup process, in addition to the optional attestation process which ensures both client and OS that the `Bento` node is patched and up-to-date. This additional communication does not alter or affect anything with the underlying Tor protocol, as all information is just passed as data through established Tor circuits to the `Bento` node.

4 Security Analysis

In this section, we discuss the security implications of our proposal and argue that it inherits strong guarantees of correctness, confidentiality, and integrity from its design.

`Bento`-side computation model. The client and onion service send their circuits to the `DeTor`$_{OS}$ function running on a `Bento` server. By the `Bento` design,

[5] That has been provisioned with a TLS certificate as part of the `Bento` setup.

the $\texttt{DeTor}_{OS}$ function is executed within a TEE, and the server is, therefore, unable to learn the circuit information provided (or even the result of the calculation). And as the client only receives the result of the calculation, neither the onion service nor the client learn anything new about each other beyond the computation output. In other words, the client and the onion service do not compromise their anonymity by participating in $\texttt{DeTor}_{OS}$ (beyond the obvious and unavoidable fact that in never-once the client learns that the onion service cannot be in the forbidden region).

It is worth noting that a curious client could try to use this fact to attempt to locate an onion service. Through numerous never-once queries with different forbidden regions, a client can attempt to learn which regions an onion service might be near based on what regions it cannot avoid. To thwart such an attack, one could envision extending the never-once function in such a way that it is able to let an onion service know if a single client has made numerous never-once queries about it; the onion service could then choose not to participate in future never-once queries to protect its privacy. We leave such an extension for future work at this time, but discuss potential solutions in more depth in Section 6.

Local computation model. In our second scenario, much of the sensitive computation is done on the client and onion service respectively, which allows all involved parties to never need to export their circuit information to anyone. For never-once avoidance, the client and the onion service calculate never-once on their own circuits and send only the result of this computation (a boolean which denotes whether avoidance was achieved or not) to the `Bento` server. Thus even if the `Bento` server was not using a TEE, it only learns the boolean values and the result of the AND. This decreases the level of trust required in the $\texttt{DeTor}_{OS}$ function, but comes at a cost of increased computation for both parties involved. In never-twice avoidance, even though much of the computation is done client-side on the respective circuits, we still rely on the `Bento` server to compute the intersection of the countries and return the result. Thus we again lean on the fact that our $\texttt{DeTor}_{OS}$ function will be running in a TEE, protecting the confidentiality of any data.

Integrity and correctness. The final important properties that we must guarantee are integrity of the data and correctness of the avoidance computation. We again rely heavily on the guarantees provided to us by the programmable `Bento` architecture. Because $\texttt{DeTor}_{OS}$ is running within a TEE on a `Bento` node, data (such as circuit and relay information) is protected from any tampering by the middlebox operator. This model also allows both the client and onion service to be assured of the correctness of the computation on the given data, as the $\texttt{DeTor}_{OS}$ function must be correctly executed. Additionally, we must ensure the correctness of the inputs to the avoidance computation. As both the client and OS have established a TLS connection that terminates inside the enclave where the $\texttt{DeTor}_{OS}$ function is running, this provides a secure channel for both to transfer information to $\texttt{DeTor}_{OS}$ while preventing tampering by the node operator.

5 Evaluation

In this section, we present the evaluation of $\texttt{DeTor}_{OS}$ for both never-once and never-twice avoidance. We aim to show that these techniques are feasible for 6-hop circuits, that is, that even with these provable avoidance techniques in place, a client still is able to (easily) find a circuit to connect to the onion service. For both never-once and never-twice avoidance, we use the same experimental setup and dataset as Li et al. [17], i.e., choosing our source-destination pairs from the Ting set of 50 relays and utilizing their latency measurements [4], but we do so for circuits with 6 hops to replicate the connection to a Tor onion service. Because adding three extra hops to a circuit exponentially increases the number of possible circuits to test[6], we elect to randomly sample one million circuits per source-destination pair (where each end of these pairs resides in a different country) rather than evaluating every possible circuit. We assume that `Bento` nodes have roughly the same geographic distribution as regular Tor relays.

5.1 Never-Once

We evaluate how successful $\texttt{DeTor}_{OS}$ is at avoiding various regions around the world, using a δ of 0.5 (recall that δ is a user configurable value $0 \leq \delta \leq 1$ where the higher the δ, the fewer potential compliant circuits will exist because of the higher burden of proof of avoidance). We do so by considering eight countries that are either very prominent for being on common routes, have many Tor relays, or are known to practice censorship, and comparing their success rates for the source-destination pair. Note that except for China, Japan, and North Korea, our results (i.e. success/failure) are proportional to [17], even if the overall success rates are slightly lower due to the 6-hop architecture.

We present our success rates in Figure 3. Each bar represents, for one of the aforementioned forbidden regions, the fraction of source-destination pairs that: successfully avoid the forbidden region over at least one circuit (*green*); terminate in the forbidden region and thus cannot achieve provable avoidance (*black*); and circuits that fail provable avoidance with real RTTs (although they theoretically avoid the forbidden region) (*red*).

Overall, our success rates are slightly lower than for DeTor's original evaluation over three-hop circuits. This is to be expected though. In the onion service setting, we are traversing six hops, and adding more hops in the circuit increases the chances that it will cross a forbidden region. Additionally, because of the extra time it takes for a packet to cross the six hop circuit, we are less certain of a circuit's ability to avoid the forbidden region. The fact that we are able to achieve even modest success rates for many forbidden regions is surprisingly positive. It is also worth noting that as we randomly sampled circuits to achieve a feasible experimental setup, this dataset is a small fraction of the actual Tor

[6] Since there are 50 possible Tor relays in the dataset and we choose 6 without replacement, this gives us over 36 billion circuits, which was infeasible to evaluate for never-once.

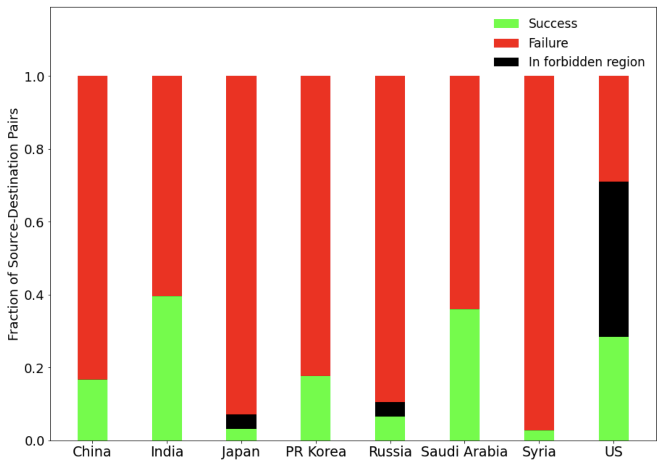


Fig. 3. Success of $\texttt{DeTor}_{OS}$ at never-once avoidance of various forbidden regions. Each bar represents the avoidance rate of the given country for the one million sampled source-destination pairs.

relays that are deployed today. Because the overall Tor network is denser and has a larger diversity of hosts, we anticipate that $\texttt{DeTor}_{OS}$ will actually perform much better in practice.

We also note that the Ting dataset is modeled based on the real configuration of the Tor network (albeit a slightly older configuration). This, of course, means that a large portion of this dataset has relays that reside in locations like the United States and Europe. As a result of this, it is difficult to, for example, find circuits that avoid the United States. However, it is worth noting that of the circuits where the entry and exit node are not in the United States, $\texttt{DeTor}_{OS}$ avoids the United States around 50% of the time, which is quite encouraging.

5.2 Never-Twice

We evaluate how successful $\texttt{DeTor}_{OS}$ is at never-twice avoidance by generating candidate circuits for our one million previously sampled source-destination pairs from the Ting dataset [4] where both the source and destination are in different countries, as never-twice is impossible when the source and destination nodes are in the same country.

We then sample 1000 circuits from each source-destination pair and see if they can provide a proof of never-twice avoidance. Doing this showed that 72.4% of our sampled source-destination pairs have a successful proof of never-twice avoidance. While this number is very encouraging, it too is lower than in DeTor's original three-hop experiments (which achieved about 98% success rates for a similar never-twice avoidance experiment). This, too, is expected; with the extra three nodes added to a circuit, if the ellipses' of a circuit's entry and exit legs go through at least one common country, the added round trip time due to various network factors increases the difficulty of providing a proof of avoidance. We also hypothesize that we would have better performance if deployed on the live Tor network as the set of clients and destinations are exponentially greater than the combinations within the dataset.

5.3 Performance

We finish our evaluation by briefly discussing the performance of $\texttt{DeTor}_{OS}$ and its potential impacts on latency.

The use of $\texttt{DeTor}_{OS}$ will add additional connection establishment latency for a user who wishes to run it before they connect to an onion service. To test this, we ran our $\texttt{DeTor}_{OS}$ function ten times on a `Bento` node running in the US, using randomly generated circuits with actual Tor relays, with a client located in the US and an OS located in Germany. On average, it took 64.85 seconds for our function to compute the never-once avoidance proof. While this time is not insignificant, as our function must take various network timing measurements for six Tor relays and then also compute the avoidance proof, we note that a user will only need to run this once for a specific circuit/OS pair (and that it often takes this long to access an onion service itself even without these computations). Besides the additional computational overhead incurred by using $\texttt{DeTor}_{OS}$ to verify never-once or never-twice for both the onion service and the client, there is next to no additional latency involved. In fact, as observed in [17], because circuits with a lower round trip time are more likely to be $\texttt{DeTor}_{OS}$ compliant, there is likely less latency than if the client were to use a Tor generated circuit to connect to the onion service.

A potential source of additional performance overhead (and hence latency) is the use of `Bento` (and hence conclaves and SGX) to realize $\texttt{DeTor}_{OS}$. We note that the overhead induced by this should be nominal on $\texttt{DeTor}_{OS}$ itself. SGX runs computations at essentially native speed, which means that it has little effect on the performance of $\texttt{DeTor}_{OS}$ computations. The largest overhead incurred for this model is context switching, and a comprehensive analysis of conclaves and SGX overhead in [7] demonstrated that this overhead was reasonable for even a CDN-like latency sensitive application[7]. As such, we believe that such minimal overhead should not be impactful or add to the overall latency induced by $\texttt{DeTor}_{OS}$.

6 Discussion and Future Work

Provable geographic avoidance for onion services was once thought to be impossible, since no one entity was able to safely know and evaluate every hop on the path. We have demonstrated that through the application of secure,

[7] Given this, we do not repeat similar experiments here and instead refer the interested reader to [7] and [22] for more information.

programmable middleboxes, provable avoidance is possible and surprisingly effective. We believe this opens up several interesting and immediate avenues for future work.

First, our current protocol only operates under the honest-but-curious model, assuming that the onion service correctly reports its path to the `DeTor`$_{OS}$ function and does not lie to the client or try to actively subvert the protocol in some way. While in practice there are likely large numbers of honest onion services that are deployed to benefit users and will follow such a protocol, and we believe it is valuable to demonstrate that geographical avoidance is possible at all with onion services, we also desire our geographical avoidance protocol to work in the face of active adversarial involvement. While this could be trivially addressed by also requiring a TEE on the onion service side that could directly communicate with the `Bento` function, this is a strong assumption that we would like to avoid. Without the use of TEEs, this seems like a challenging problem to address, and one that might involve inherent changes to the underlying `DeTor`$_{OS}$ protocol and computations.

Second, we inherit the use of TEEs from the design of `Bento` and rely on them to ensure the privacy and correctness of the computations. While we have seen a number of attacks on TEEs thus far, we have also seen TEE vendors provide patches and updates for all such attacks, and remote attestation mechanisms allow for users to gain assurance that a computer is fully patched against all known vulnerabilities. However, we still briefly discuss the impacts of a TEE compromise on `DeTor`$_{OS}$. Since we rely on the TEE for both correctness and confidentiality, a breach would likely harm both of these properties, resulting in the potential leakage of circuit information to the node operator and a weakening of the correctness guarantees of the computations. This is where the difference in the `Bento`-side versus local computation model can be beneficial, as the information leakage can be minimized with local computation (though we still lose strong correctness guarantees on the returned result). As such, an interesting avenue of future work would be to explore mechanisms, such as multi-party computation, that would allow us to still leverage the idea of programmable anonymity networks, without relying on the need for TEEs.

Third, as we discussed briefly in Section 4, our current protocol does not protect an onion service from a malicious client that wishes to try to deanonymize it through repeated queries about avoidance of distinct geographic regions. We envision that one simple way to mitigate this would be to extend the `DeTor`$_{OS}$ function to track the number of times a user invokes it with regards to a specific onion service, and either rate-limit queries or notify the onion service of repeated queries, allowing it to decide whether to participate in the protocol or now. One way to achieve this rate-limiting in a privacy-preserving manner would be through issuing k-show anonymous credentials [6,24]. A client wishing to visit an onion service would then first obtain an anonymous credential (which refreshes every day) from the issuer (which could be a `Bento` server). The client would then show this anonymous token every time she wants to execute `DeTor`$_{OS}$. This limits the client's access to the onion service since the client can execute `DeTor`$_{OS}$

only k times per day. However, as the `DeTor`$_{OS}$ function is user-controlled, the onion service itself would also need to, through the `Bento` attestation process or other mechanism, ensure itself that the deployed function it is interacting with contains these protections. Another interesting piece of future work would be to investigate if there are other avenues to thwart such an attack.

Fourth, while the results of both never-once and never-twice for `DeTor`$_{OS}$ are promising, it is critical to come up with ways to reduce the additional latency added by adding the three extra hops required to connect to a hidden service. It is also imperative to find new ways to speed up the calculations that `DeTor`$_{OS}$ (and the original detor paper [17]) use in order to reduce the computational overhead required.

Finally, taking a step back, `Bento`'s programmable middleboxes made provable avoidance possible by outsourcing a sensitive computation to a mutually trusted third party. We wonder: what other services could be run in a similar fashion? Perhaps it is possible to build disaggregated services on top of a programmable anonymity network by disseminating pieces of code across the network, so that even if one part of it is compromised other parts can replicate and recover. Perhaps it is possible to randomize where any computation in the network occurs, so that the onion service is hidden even from the user who is running it. Our hope is that this work spurs such considerations, and to assist in future work we have made our code publicly available[8].

7 Ethical Considerations

All of the data in our experiments comprised only our own traffic: not any actual users' data. Our never-once performance evaluation was performed on the actual Tor network. However, this only involved collecting latency times for various nodes and circuits on the network, and data was gathered in a rate-limited fashion to ensure that our experiments would not impact the performance of the Tor network. Also, we deployed both our own Tor node and `Bento` node, which was limited to only our own traffic so as not to affect the larger Tor network.

8 Conclusion

In this work we present `DeTor`$_{OS}$, the first technique that is able to provide provable geographic avoidance for onion services. We achieve this by leveraging recent advances in programmable anonymity networks, which allow the user and onion service to jointly compute on their circuits, without leaking information to the other party. While we implement this work primarily with the `Bento` architecture, we believe the overall design of `DeTor`$_{OS}$ can be used with any architecture that supports such programmability in anonymity networks [23]. We showed that our design and implementation is able to achieve never-once and never-twice avoidance at rates that are encouraging. We also discuss a number of avenues of future work that this first deployment opens up.

[8] `https://bento.cs.umd.edu`

Acknowledgments

We thank the anonymous reviewers for their helpful comments. Arushi Arora and Christina Garman's work was partially supported by NSF grant CNS-1816422. Dave Levin's work was partially supported by NSF grant CNS-1943240.

References

1. Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.
2. ARM security technology: Building a secure system using TrustZone technology. `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf`.
3. Daniel Arp, Fabian Yamaguchi, and Konrad Rieck. Torben: A practical side-channel attack for deanonymizing tor communication. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 597–602, 2015.
4. Frank Cangialosi, Dave Levin, and Neil Spring. Ting: Measuring and exploiting latencies between all tor nodes. In *Proceedings of the 2015 Internet Measurement Conference*, pages 289–302, 2015.
5. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, 2004.
6. Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. *Cryptology ePrint Archive*, 2013.
7. Stephen Herwig, Christina Garman, and Dave Levin. Achieving keyless cdns with conclaves. In *USENIX Security Symposium*, 2020.
8. Intel. L1 Terminal Fault, 2018. `https://software.intel.com/content/www/us/en/develop/articles/software-security-guidance/advisory-guidance/l1-terminal-fault.html`.
9. Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. Technical report, Office of Naval Research Arlington VA, 2014.
10. Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 337–348, 2013.
11. Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel Software Guard Extensions: EPID Provisioning and Attestation Services, 2016.
12. Ishan Karunanayake, Nadeem Ahmed, Robert Malaney, Rafiqul Islam, and Sanjay Jha. Anonymity with tor: A survey on tor attacks. *arXiv preprint arXiv:2009.13018*, 2020.
13. Katharina Kohls, Kai Jansen, David Rupprecht, Thorsten Holz, and Christina Pöpper. On the challenges of geographical avoidance for tor. In *Network and Distributed System Security Symposium (NDSS)*, 2019.
14. Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security Symposium*, 2015.

15. Dave Levin, Youndo Lee, Luke Valenta, Zhihao Li, Victoria Lai, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. Alibi routing. *ACM SIGCOMM Computer Communication Review*, 2015.
16. Philip Levis. The collateral damage of internet censorship by dns injection. *ACM SIGCOMM CCR*, 42(3):10–1145, 2012.
17. Zhihao Li, Stephen Herwig, and Dave Levin. Detor: Provably avoiding geographic regions in tor. In *USENIX Security Symposium*, 2017.
18. Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.
19. Hovership Nebuchadnezzar. The collateral damage of internet censorship by dns injection. *ACM SIGCOMM CCR*, 42(3):10–1145, 2012.
20. Lasse Overlier and Paul Syverson. Locating hidden servers. In *IEEE Symposium on Security and Privacy*, 2006.
21. The Tor Project. Tor Manual, 2022. `https://2019.www.torproject.org/docs/tor-manual.html.en`.
22. Michael Reininger, Arushi Arora, Stephen Herwig, Nicholas Francino, Jayson Hurst, Christina Garman, and Dave Levin. Bento: Safely bringing network function virtualization to tor. In *ACM SIGCOMM*, 2021.
23. Florentin Rochet, Olivier Bonaventure, and Olivier Pereira. Flexible anonymous network. *arXiv preprint arXiv:1906.11520*, 2019.
24. Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. *Cryptology ePrint Archive*, 2022.
25. Matthew J Ryan, Morshed Chowdhury, Frank Jiang, and Robin Doss. Avoiding geographic regions in tor. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020.
26. Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. Routing around decoys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 85–96, 2012.
27. Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security Symposium*, 2014.