On the Routing-Aware Peering against Network-Eclipse Attacks in Bitcoin

Muoi Tran
National University of Singapore

Akshaye Shenoi
National University of Singapore

Min Suk Kang*
KAIST

Abstract

Safeguarding blockchain peer-to-peer (P2P) networks is more critical than ever in light of recent network attacks. Bitcoin has been successfully handling traditional Sybil and eclipse attacks; however, a recent Erebus attack [57] shows that effectively eclipsing a Bitcoin node is possible when the attack is combined with a network-Sybil capability; i.e., a malicious transit network can create millions or more Sybil identities. Given the immediate availability and stealthiness of the Erebus attack, Bitcoin Core has quickly implemented a few simple protocol/parameter changes to mitigate it. Our large-scale evaluations of these quick patches and three similar carefully-designed protocol tweaks confirm that, unfortunately, no simple solution can effectively handle the attack. This paper focuses on a more fundamental solution called routing-aware peering (or RAP), a proven silver bullet in detecting and circumventing similar network adversaries in other P2P networks. However, we show that, contrary to our expectation, preventing the Erebus attacks with RAP is only wishful thinking. We discover that Erebus adversaries can exploit a tiny portion of route inference errors in any RAP implementations, which gives an asymmetric advantage to the network adversaries and renders all RAP approaches ineffective. To that end, we propose an integrated defense framework that composes the available simple protocol tweaks and RAP implementation. In particular, we show that a highly customizable defense profile is required for individual Bitcoin nodes because RAP's efficacy depends significantly on where a Bitcoin node is located on the Internet topology. We present an algorithm that outputs a custom optimal defense profile that prevents most of Erebus attacks from the top-100 large transit networks.

1 Introduction

Many blockchains run on *permissionless* decentralized peer-to-peer (P2P) networks. Their openness and decentralization have been the keys to the success of cryptocurrencies and smart contracts; however, this comes with a price — they are potentially vulnerable to Sybil [15] and eclipse [51] attacks. Bitcoin [40], one of the most popular blockchain systems, has been effectively addressing Sybil and eclipse attacks. First, a Sybil attacker in Bitcoin can create many peer identities with different IP/port combinations using a single peer machine. In this way, the attacker can act like multiple peers in the system. However, the spawned Sybil identities do not cause much harm because the Bitcoin P2P protocol groups all the Sybil identities from a single IP address and contains them. Second, eclipse attacks target a specific node and manipulate the target's peering algorithm to isolate it from the rest of the network. The current Bitcoin client software is robust to the known attack [27] after patching a few vulnerabilities in its peering algorithm [25, 26].

However, a more recent attack, dubbed Erebus [57], shows that Bitcoin is still vulnerable to a *persistent* eclipse attack with the *network-Sybil capability*; i.e., a malicious autonomous system (AS) creating large numbers of peer identities via IP spoofing. Bitcoin's permissionless nature allows any sizable network adversaries (e.g., the top-100 largest ASes [9]) to easily create and use massive Sybil identities and ultimately control all of a target node's connections. Bitcoin Core has implemented a few quick patches to mitigate this attack [41, 52] with the caveat that none of them are complete solutions yet [21]. Our evaluations (see Section 3) confirm that these simple protocol fixes (and some other simple protocol tweaks) only marginally increase the Erebus attack execution time.

A more desirable long-term approach to the Erebus attack is to *remove* the network adversary's capability of utilizing large numbers of Sybil identities. One recurring suggestion for this is what we call a *routing-aware peering* (or RAP) approach. This option is a proven silver bullet in detecting and circumventing similar network adversaries in Tor P2P networks [1, 45, 53]. RAP empowers individual Bitcoin nodes to identify Sybil peer identities by analyzing the routes towards all potential peers. Some early discussions on RAP

*Corresponding author.

approaches in Bitcoin have been initiated, but no concrete implementation is available, possibly due to higher implementation complexity than quick patches.

This paper argues that preventing Erebus attacks with RAP is, unfortunately, wishful thinking. Unlike RAP in similar Tor P2P networks, RAP implementations in Bitcoin fail to prevent network adversaries and only increase the attack execution time slightly. We find that the critical weakness of all possible RAP approaches in Bitcoin is their infrequent yet inevitable route inference errors; e.g., less than 6% of miss rate. In a permissionless Bitcoin P2P network (unlike the semi-permissionless Tor network), such a low route inference error rate in RAP can be translated into tens or even hundreds of thousands of Sybil identities, which is still enough for a successful Erebus attack. By exploiting these rare inference errors in RAP, adversaries obtain an *asymmetric advantage* over a target Bitcoin node and successfully eclipse it in most cases.

After learning that no single countermeasure in the P2P layer can effectively mitigate the Erebus attack, we alternatively aim to design a defense system that integrates all these available countermeasures to provide a workable, practical solution against the attacks. Integrating the simple patches and RAP implementations may first seem obvious. However, it is not straightforward for two reasons: (1) simply implementing more of the available countermeasures in a Bitcoin node may not necessarily guarantee better defense performance as the security against the Erebus attack is *non-monotonic*; and (2) there exists no one-size-fits-all RAP operation in practice because its performance and cost vary drastically depending on *where* on the Internet topology a Bitcoin node is located. To that end, we present a new customizable defense framework for Bitcoin node operators to make an informed decision regarding the Erebus mitigation, particularly depending on their nodes' locations on the Internet topology. We also present an efficient algorithm that outputs RAP's optimal operating point along with the simple countermeasures, which prevents up to 98% of Erebus attacks from the top-100 large transit networks.

We summarize our contributions as follows:

- We evaluate a set of six (three old and three new) simple protocol/parameter tweaks against a recently proposed Erebus attack [57] using (and improving) a publicly available evaluation framework (§3). We conclude that none of them successfully mitigate the Erebus attack, unfortunately.
- We then evaluate a highly promising defense approach called routing-aware peering (RAP) and show that, contrary to our expectation, no practical RAP implementations for Bitcoin can prevent the Erebus attacks. They can only slightly increase the attack execution time (§4 and §5). Our new finding is that a minimal route inference error rate (which is shown to cause no significant impact to the previous defenses in similar P2P attacks) can give an asymmetric advantage to the Erebus network adversaries.
- We propose an integrated defense framework by composing the available simple protocol patches and the RAP implementations (§6) to protect Bitcoin nodes against the majority of the Erebus attacks. Our framework allows highly customizable defense profiles for Bitcoin nodes in diverse network environments that balance the desired level of robustness against the Erebus attack and the costs of RAP operation.

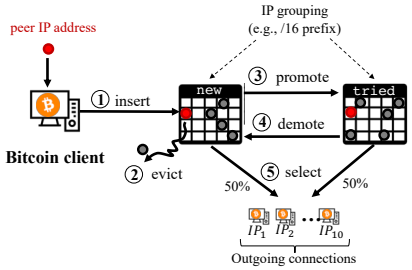


Figure 1: An illustration of the *life cycle* of peer IP addresses in Bitcoin. IPs are stored in two-tier tables where reliably reachable IPs are selected to be outgoing peers, and "terrible" (e.g., older than 30 days) IPs are eventually evicted.

2 Threat Model

In this paper, we consider the same attack goals and capabilities of the original Erebus attack [57]. As a background, we briefly introduce the Bitcoin peer-to-peer network (§2.1), and then we describe the Erebus attack model (§2.2).

2.1 Background on the Bitcoin P2P Protocol

Bitcoin [40] relies on a P2P network of individual nodes to maintain a replicated distributed ledger, the *blockchain*, which stores the historical ownership information of all funds in the system. A fund can be transferred from one user to another in a transaction. Valid transactions are grouped into a block, and the blockchain is periodically extended with a new block via a mining process. All transaction and block information is propagated by the P2P network until all nodes in the system are synchronized with the same state. Since Bitcoin is permissionless (i.e., nodes can freely join and leave the system), it allows a node to notify its existence to the network via self-advertising its peer identity (e.g., IP address) to some peers, who then propagate the address information to all other nodes. Bitcoin aims to form a random P2P network, where each node establishes ten outgoing connections that are carefully selected from its local database that stores other peers' identities [13]. Nodes with public IPs also accept up to 115 incoming connections and are considered reachable.

For an easier understanding of how the peer information (i.e., IP addresses) is maintained and used in Bitcoin, let us

zoom into the internal workings of a single Bitcoin node and show the *life cycle* of an IP address, i.e., all possible steps that it would experience within the internal data structure of the node in Figure 1. The internal peer database of a node consists of two tables: the `new` table stores newly-propagated IPs, and the `tried` table stores the IPs that have been connected to. Both tables manage IPs in groups (e.g., /16 prefix for IPv4) where each group can occupy only a small portion of the tables; see previous work [27, 57] for the detailed description of IP allocation. There are five steps in the life cycle of an IP address:

- In Step ①, the IP address propagated by other peers is initially inserted into the `new` table.
- If the new IP is inserted into an already occupied slot and the existing IP is "terrible" (e.g., older than 30 days), the existing one is evicted from the table; see Step ②. We observe that most IPs in the `new` table tend to become unreachable in the long run.
- Every two minutes, one randomly-chosen IP address from the `new` table is tested by a short-lived connection (i.e., feeler connection) and is promoted to the `tried` table if it is reachable; see Step ③. From our observation, most IPs in the `tried` table tend to be reachable even after a few weeks later.
- Step ④ describes the demotion of an IP address from the `tried` table to the `new` table if another IP is inserted into its slot and it is tested to be unreachable.
- In Step ⑤, when there are not yet ten outgoing connections, a reachable IP is selected at random from either `new` or `tried` table (which table is chosen is also randomized) to establish a connection. If this new outgoing peer is selected from the `new` table, it is also promoted to the `tried` table as in Step ③.

2.2 The Erebus Attack

The Erebus attack was recently presented as a stealthy partitioning attack against Bitcoin P2P network [57]. At a high level, the Erebus attacker follows the common attack recipes of P2P eclipse attacks, such as [27]; that is, to gradually insert adversary-controlled identities into the peer database of a targeted Bitcoin node until all connections selected by the victim are made to those peers (i.e., the victim is isolated from the rest of the network). The main difference of the Erebus attack compared to typical eclipse attacks is that a *network* adversary creates millions or more *network-Sybil* peer identities by spoofing IP addresses of any ASes behind her network with respect to the victim node's location. Figure 2 illustrates an example of the Erebus attack in which the adversary "AS666 Evil Telecom"[1] injects selected IP addresses (e.g., *B* and *C*) into the victim's internal database in the form

[1]The AS number 666 is used as a symbol of the devil's number. It is *not* meant to indicate a real AS with AS number 666.

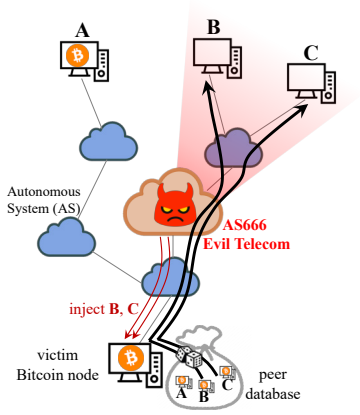


Figure 2: An illustration of the Erebus attack [57]. "AS666 Evil Telecom" inserts some Sybil IPs (e.g., B and C) into the peer database of the victim Bitcoin node so that peer connections towards these Sybil IPs can be made later at the victim's discretion. At some point in time, AS666 eventually hijacks all peer connections of the victim node.

of peer advertisements and waits for the victim to connect to *B* and *C*. When all connections are made to Sybil IPs, the attack is considered as successful. Note that the adversary AS must be on the paths from the victim to Sybil nodes (e.g., *B*, *C*) to spoof their IP addresses but *not* necessarily on the reverse-direction paths.

We consider a network adversary (e.g., Tier-1 or large Tier-2 ASes) that has the full control of any messages going through her own network. The attacker can also send only low-rate (e.g., once every few seconds) source-IP-spoofed packets. The attack goal is to occupy all the peer connections of a targeted node and thus *isolate* (or eclipse) the targeted Bitcoin node from the rest of the Bitcoin P2P network.

Assumptions for victim nodes. We make the same assumptions made in the original Erebus attack work [57] for the victim Bitcoin nodes.

- *Identified by public IPs.* We consider Bitcoin full nodes with public IP addresses. There are about 8,000 such public nodes as of February 2021 [59]. Note that SPV client nodes [40], VPN-connected nodes, or Bitcoin nodes connected via Tor are out of scope as the original Erebus attack work also does not target them [57].
- *IPv4 only.* The vast majority (e.g., 85%) of Bitcoin nodes run with IPv4 [59], and we also assume that a victim node has a single IPv4 address. We limit ourselves to the IPv4 space in this paper, considering that an IPv4 Bitcoin node can only connect directly to other IPv4 nodes [24].
- *No central regulating authority.* Network attacks at the P2P layer are often prevented with some trusted central regulating authorities; e.g., a repository of white-listed (or black-listed) peers or connecting through relay nodes [3, 18,

20]. However, relying on such a centralized party is a less ideal approach, particularly in cryptocurrencies, as it may violate the permissionless blockchain principle. Thus, in this paper, we only consider solutions without any central authorities.

- *No moving target defense.* Targeted network attacks are sometimes effectively mitigated when a victim host frequently changes its network identity [2, 30, 31]. However, we argue that moving target defenses should only be the last resort for Erebus mitigation. When a moving target defense is deployed in a permissionless decentralized P2P network, a peer node needs to rebuild its P2P connectivity at every IP change. However, frequent IP changes in Bitcoin are known to seriously damage the peers' network connectivity and negatively affect the block and transaction propagation [29].
- *No cross-layer solutions.* The Erebus attack work [57] suggests one potential countermeasure called 'smart peer eviction policy' that requires some interactions between the consensus layer and the P2P network layer. Despite its potentials, such a cross-layer solution would unavoidably complicate the implementation and, worse, may open up new vulnerabilities. Thus, in this paper, we strictly limit ourselves to the P2P network-layer only solutions.

3 Limitations of Simple Countermeasures

Considering its immediate availability and the stealthiness property of the Erebus attack, the Bitcoin Core team has announced two simple patches not long after the publication of the Erebus attack [41, 52]. However, no systematic, large-scale evaluations have been conducted on such simple countermeasures. As a first step towards the Erebus mitigation, we implement and evaluate several simple countermeasures (including the two already adopted quick patches) to the Erebus attack. We consider a countermeasure is *simple* when the required change to the Bitcoin codebase is only a few lines of source code or even a single parameter change.

We first review three proposed tweaks from existing work and present three additional tweaks derived from our IP "life cycle" in Bitcoin P2P (§3.1). We then empirically evaluate the Erebus attacks against them and confirm that tweaking the Bitcoin protocol is insufficient to address the attacks (§3.2).

3.1 Bitcoin Protocol Tweaks

We present several protocol tweaks along with their descriptions, objectives, caveats, and deployment status in Table 1.

Previously-proposed simple countermeasures. The original Erebus paper [57] suggests four changes to the Bitcoin protocol, i.e., *ASN-based grouping*, *more outgoing connections*, *table size reduction*, and *smart peer eviction policy*, and we implement and test the first three countermeasures. As of this writing, the ASN-based grouping (**T1**) is supported in Bitcoin Core as an experimental feature [13, 41]. Since Bitcoin 0.19.0, two more outgoing connections are added for propagating only the block data, which happens infrequently (e.g., few MBs every ten minutes) and unlikely creates the traffic burden to the network. If the additional connections are block-relay-only or the on-going developments of bandwidth-saving for Bitcoin transactions (e.g., Erlay [42]) are integrated into Bitcoin, adding even more connections (**T3**) is worth considering. The table size reduction tweak has not been developed, perhaps because it contradicts the protocol tweak made after the Eclipse attack [27] where both `new` and `tried` tables are increased by four times.

The Eclipse paper [27] makes several suggestions to Bitcoin and most of them have been deployed over the years, except the *anchor connection* tweak (**T2**) that resurfaces after the Erebus attack is presented [52]. Currently, Bitcoin preserves two block-relay-only connections (i.e., anchors) when rebooting to prevent eclipse attackers from terminating existing (likely legitimate) connections [13].

Additional simple countermeasures. An important observation from the "life cycle" of Bitcoin peer IP addresses (see Section 2.1) is that when the Erebus attack occurs, the attacker IPs can easily dominate the `new` table but cannot evict legitimate reachable IPs from the `tried` table. This suggests that if we increase the legitimate IPs in `tried` table or prioritize IPs from `tried` table for peer selection, the chance for selecting the attacker's peers will be significantly lowered.

Taking this observation as the guiding light, we find *three new* tweaks, i.e., *always select IPs from* `tried` *table*, `tried` *table reduction*, and *feeler interval reduction*. First, in Step ⑤ in Figure 1, when selecting an IP address for an outgoing connection, Bitcoin nodes should always select from the `tried` table (if `tried` table has sufficient IPs) (**T4**), instead of randomly selecting from either `new` or `tried` tables. This requires the attacker to spend significantly more time waiting for IP promotion and removes the risk of selecting attacker IPs from `new` table. Second, because reachable legitimate nodes cannot be removed from `tried` table, we should reduce the `tried` table size (**T5**) so that the number of empty slots occupied by the adversary is minimized. Particularly, the `tried` table (consists of 16 thousand slots currently) should be well-aligned with the number of reachable Bitcoin nodes in the system (about 8 thousand nodes). Note that we do not suggest reducing the `new` table's size as in the original Erebus paper because the attacker can easily replace the majority of the unreachable IPs in `new` table anyway. Third, the promotion rate can be increased (i.e., by reducing the feeler connection interval) (**T6**) so that there are more legitimate reachable IPs in the `tried` table. However, this tweak may be beneficial for the attacker if there are too many empty slots because attacker IPs will also be inserted quicker.

Table 1: Bitcoin protocol tweaks. **T1–T3** are previously proposed while **T4–T6** are newly derived from the IP life cycle.

Name	Description	Objective	Caveat	Status
ASN-based grouping (T1)	IP addresses in the two tables are grouped based on their AS number, instead of prefix (/16 for IPv4 or /32 for IPv6).	To reduce attacker IPs in the tables, as they usually belong to fewer ASes than they do to prefix groups.	Effectiveness may be insignificant [21, 57].	Proposed in [57] and included in Bitcoin as a non-default, under-testing option since version 0.20.0.
Anchor connection (T2)	Upon rebooting, some last-known outgoing peers, called anchors, are re-connected.	To mitigate a common strategy of eclipse attacks [27, 57] that removes all existing connections of the victim via rebooting it.	Attacker IPs can also be selected as anchors.	Proposed in [27] and being developed [52].
More outgoing connections (T3)	The number of outgoing connections is increased.	To lower the chance of selecting all attacker IPs as outgoing peers, forcing the attacker to occupy the database with a higher ratio.	The P2P network will need to propagate more traffic.	Suggested by [27, 57]. Since Bitcoin 0.19.0, two outgoing connections are added.
Always select IPs from `tried` table (T4)	When selecting an IP address for outgoing connections, the `tried` table should always be selected.	To force the adversary to spend more time waiting for IP promotion and to remove the risk of selecting attacker IPs from the `new` table.	IPs from `new` table will still be selected when there are not many IPs in `tried` table.	Derived from step ⑤ of the IP life cycle.
`Tried` table size reduction (T5)	The `tried` table should have a smaller size so that its space is well-aligned with the number of nodes in the system.	To reduce `tried` slots occupied by the adversary-chosen IPs when the attack happens because there is less space and it is impossible to remove reachable legitimate IPs.	The size should be adjusted accordingly to the state of the network.	Derived from step ③ of the IP life cycle.
Feeler interval reduction (T6)	The interval of the feeler connections is shortened.	To increase legitimate reachable IPs in the `tried` table via IP promotion.	If there are many empty slots in the `tried` table, attacker IPs are also inserted quicker.	Derived from step steps ③, ④ of the IP life cycle.

3.2 Evaluation of Simple Protocol Tweaks

We now evaluate the discussed simple protocol tweaks.

Evaluation framework. We use the open-source Bitcoin emulator [6], which was used to evaluate the original Erebus attack [57], to emulate a Bitcoin node's peer selection. We also update the emulator to reflect the latest Bitcoin version 0.21.0 [13] with a few recent changes; e.g., two out of ten outgoing connections are used for only block data propagation. For realistic operations, we feed the emulated node with the real Bitcoin address advertisements (i.e., `addr` messages) containing the real IPs that our live Bitcoin node collected in 380 days (from November 18, 2018, to December 4, 2019). Upon making an outgoing connection to a peer, the emulated node queries the Bitnodes dataset [59] to check if the peer is reachable (i.e., it accepts incoming connections) at that moment in the simulation time. We run our experiments on a Dell PowerEdge R630 server with 40 cores of Intel(R) Xeon(R) E5-2640 v4 @ 2.40GHz and 128 GB of memory. Emulating a Bitcoin node running for 380 days on one CPU takes about 20–30 minutes on average.

Attack scenarios. We consider the 100 largest ASes in the current Internet ranked by their customer cone size [9] as the adversaries, similar to the evaluation of the Erebus attacks [57]. To learn the attacker IPs (i.e., that have the attacker on the victim-to-IP paths), we measure 47.2 *million* data-plane paths from 59 globally-distributed nodes to all 800 thousand available IPv4 prefixes in the Internet. Those include 21 nodes hosted at different regions of five popular cloud providers (i.e., Amazon, OVH, DigitalOcean, Hetzner, and Alibaba)[2], 26 PlanetLab nodes [46][3], and 12 PEERING servers [50][4]; refer to Appendix A for more details of our data-plane route measurement. Considering our measurement nodes as the victim Bitcoin nodes, we have 5,900 different attack scenarios.

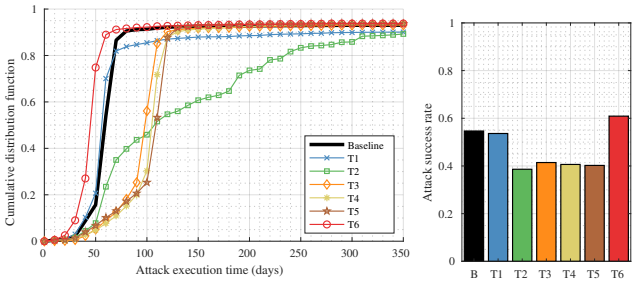
In each scenario, we execute the Erebus attack against our emulated node, which runs attack-free for 30 days before the attack commences. Then, we wait up to 380 days (i.e., our maximal realistic emulation duration) to measure the required attack execution time for controlling all outgoing connections of the victim, i.e., the attack is successful. When we want to directly estimate the attack success probability with respect to the actual IP churn rate (i.e., how often Bitcoin nodes change their IP addresses), we model the online time of the victims from the actual distribution (see the measured IP churn rate of Bitcoin nodes in the wild in Appendix B) and compute the success/failure of each attack attempt.

Tweak implementations. Here, we evaluate six tweaks **T1–T6** in Table 1. To implement **T1**, we use the Routeviews Prefix-to-AS mapping [12] to map IPs into AS numbers (see Appendix C for a more sophisticated mapping being implemented by Bitcoin Core). For **T2**, we follow the configuration of Bitcoin Core [52] to preserve two block-relay-only connections across rebooting. To test **T3**, we add six more block-relay-only connections (i.e., sixteen outgoing connections in total) so that the number of outgoing connections is doubled than before the Erebus attack. When testing **T4**, our simulated nodes select outgoing peers from both tables, then switch to selecting exclusively from the `tried` table when 25% of the `tried` slots are filled. For **T5** and **T6**, we reduce the `tried` table size and the feeler interval by four times, respectively. We also evaluate the effectiveness of the

[2] The majority of Bitcoin nodes are also running on clouds [59].

[3] We have collected all our results before PlanetLab officially closes down in May 2020.

[4] We used all 12 PEERING servers that have the full routing tables, see https://peering.ee.columbia.edu/peers/ for the list of servers.



(a) CDF of required attack time. (b) Attack success rate.

Figure 3: Cumulative distribution of the required attack execution time and the attack success rate when a Bitcoin client implements the protocol tweaks **T1–T6**.

baseline client without any tweak, called **Baseline** or **B**. We use conservative parameters for **T2–T6** in our evaluations. More discussions in the community might be needed before the actual development of these tweaks.

Results. We present the defense effectiveness of the tweaks in Figure 3 with the CDF plots of the required attack duration in all attack scenarios and the rate of successful attack instances. Figure 3a shows that individual tweaks of Bitcoin protocols do not demonstrate significant improvements as they reduce the number of attack instances that require less than 350 days by only less than five percentage points. The results shown in Figure 3b confirm that tweaks **T1–T5** reduce the attack success rates insignificantly, around only 2–16 percentage points from 54.7%. Exceptionally, **T6** makes Bitcoin clients perform even worse than the baseline version, perhaps because there are too many empty `tried` slots for attacker IPs to occupy when the attacks commence.

To sum up, simple tweaks for the Bitcoin protocol are indeed beneficial for mitigating the Erebus attacks; yet, their effectiveness is only marginal, if not negative. The most robust tweak **T2** (i.e., anchor connections) is still insufficient to mitigate the Erebus attacks with the success rate of 38.6%, which urges us to look for more complex and potentially more effective solutions.

4 Routing-Aware Peering: A Rescue to Save Bitcoin from the Erebus Attacks?

Understanding the shortcomings of the quick, simple solutions, we turn our attention to a long-term solution. We aim to remove the network Sybil capability of the Erebus adversaries, which makes the attack possible in the first place. A potential countermeasure, called routing-aware peering (RAP), lets a Bitcoin node use the knowledge of end-to-end routes of all its peer connections and tries to prevent them from going through a suspicious AS, effectively disabling the attackers' network-Sybil capability. The idea of RAP has already been mentioned multiple times as a promising solution to the Erebus attack [4, 57] since it is proven effective in preventing similar network adversaries in Tor P2P networks.

This section first explains the rationale for integrating the routing knowledge into the Bitcoin peer selection for Erebus mitigation (§4.1). Then, we outline the high-level design of a RAP defense in the current Bitcoin (§4.2).

4.1 Why is RAP Believed to Prevent the Erebus Attacks?

The rationale behind RAP as a mitigation to the Erebus attacks is to empower each Bitcoin node with the *symmetric* defense capability. In the same way that an Erebus adversary exploits the end-to-end routing knowledge to place itself in a man-in-the-middle position, a target Bitcoin node can also utilize the same knowledge to detect when an attack occurs. If a victim node is also equipped with end-to-end routing knowledge and aware of which ASes are (and will be) located on the existing (and the future) peer connections, it can detect an Erebus attack campaign even before all the peer connections are made through a malicious AS. For example, as shown in Figure 2, a victim node can learn that all of its peer connections cross the evil AS666 and then try to find some new peers whose victim-to-peer routes do not include the malicious AS.

In fact, the idea of RAP defenses has been already shown to be highly effective in a different context, particularly for securing Tor [1, 45, 53] against AS-based traffic analysis attacks [19, 39, 54]. In these Tor attacks, a malicious AS aims to be on the man-in-the-middle position of many (if not all) of the Tor paths from/to the victim Tor clients and the covert public servers (e.g., Tor-client-to-entry-relay paths, exit-relay-to-server paths). The required defense capability against these Tor attacks is similar to what we aim to achieve via RAP in Bitcoin; that is, an end client learns the intermediate ASes of their Tor paths, detects these malicious-AS attacks, and chooses other Tor relays that ensure attacker-free inter-domain routes. Based on the highly promising track records of RAP-based defenses against malicious-AS attacks in Tor P2P networks, it is easily believed that RAP would also effectively mitigate the Erebus attack in Bitcoin.

4.2 Design Overview of RAP in Bitcoin

As we design a practical RAP defense logic in Bitcoin, we consider a *general* defense scenario where a Bitcoin node does *not* know the Erebus adversary AS a priori. That is, a Bitcoin node operates a RAP defense to prevent *any* intermediate ASes from overseeing all its peer connections without knowing exactly which AS is a malicious AS.[5] Note that this is a conservative defense scenario as it is strictly easier to operate a RAP defense when the malicious Erebus AS is known to the victim node.

[5] Collaborative Erebus attacks, where two or more ASes collaborate to hijack the victim's peer connections, are out of the scope of this work, as is the case in the original Erebus paper [57].

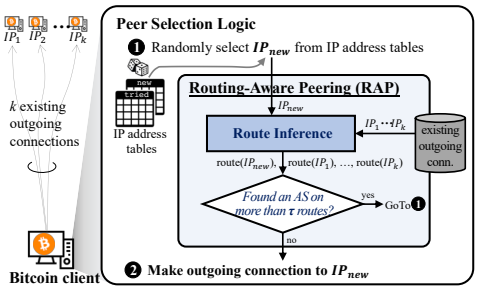


Figure 4: Bitcoin's peer selection logic and routing-aware peering (RAP) improvement. When IP_{new} is chosen for a new outgoing connection, the RAP function checks whether a malicious AS will likely be on the path.

The routing-aware peering (RAP) defense requires each Bitcoin node to obtain the routing knowledge (i.e., the inter-domain routes) for all of its peering connections. We call this new core functionality the *route inference* logic, and we add it to the existing Bitcoin's peer selection algorithm. Essentially, the route inference returns an estimated AS path from the client (i.e., itself) to any given IP address.

Figure 4 illustrates how the current Bitcoin's peer selection logic is implemented and how it can be augmented for RAP. When a Bitcoin client wishes to make one more outgoing connection, it selects an IP address IP_{new} from its internal tables at random; see step ❶. The current implementation immediately attempts to make a new outgoing connection to IP_{new}; see step ❷. Instead, the RAP implementation takes the chosen IP_{new} and checks if the new connection will likely include any potentially malicious AS before opening a connection to it. Using the route inference logic, RAP obtains the inferred routes to IP_{new} and to all existing outgoing peer IPs (i.e., $IP_1, \cdots, IP_k$). If a potentially malicious AS appears in more than τ routes, RAP rejects this new IP_{new}; otherwise, it proceeds to make an outgoing connection to it.

The threshold τ is an adjustable parameter defining the maximum number of allowed connections that share a potentially malicious AS. It controls how strictly the RAP function is operated. When τ is set to a low value, not many peer connections can share the same intermediate ASes; thus, more route diversity among peer connections is expected. On the contrary, if τ is set to a high value, the node allows multiple of its peering connections to share the same intermediate AS. Note that $\tau = 1$ forces all the peering routes to be disjoint and $\tau = 10$ disables RAP. An individual Bitcoin node can easily adjust the value of τ, and the effect of different values of τ is analyzed later in Section 6.2.

5 Why RAP Fails to Prevent the Erebus Attacks

We have explained why RAP is believed to be effective prevention of the Erebus attacks. However, our discussion in Section 4 conveniently ignores the implementation details of the route inference logic, which is the core component of any Bitcoin RAP implementations. We first review and evaluate several implementation choices for the route inference logic in RAP and show that there *must* inevitably exist some route inference error cases (§5.1). Then, we introduce a subtle but powerful Erebus attack strategy that exploits even a tiny portion of route inference errors (§5.2). The exploitation gives an *asymmetric advantage* to the Erebus adversaries, allowing them to successfully isolate (or eclipse) the targeted Bitcoin node *with* a RAP defense for the majority of cases (§5.3).

5.1 The Devil Is in the Detail: Non-idealities of Route Inference in RAP

We exhaustively list the possible ways of route inference in RAP and quantitatively compare them with our large-scale experiments in the Internet.

We first divide all feasible approaches (to the best of our knowledge) into three categories, i.e., (1) control-plane estimation, (2) control-plane look-up, and (3) data-plane measurement, and discuss their pros and cons.

(1) *Control-plane estimation:* A Bitcoin node locally computes the estimated inter-domain route for a given destination address. Based on the publicly available data (e.g., AS-level topology [10], BGP feeds [44,49]), there have been several proposals and algorithms on inter-domain route estimation (e.g., [1,37,47]). *(Pros)* Individual nodes can run the full route computation locally, and pre-computation can also be done to remove the on-line computation burden. *(Cons)* Estimation algorithms are imperfect; thus, the BGP route inference may be prone to estimation errors.

(2) *Control-plane look-up:* A Bitcoin node directly learns the BGP routes to a specific destination from the routing table (e.g., RIB) of its local BGP gateway. Unlike the previous BGP estimation, this results from the *actual* BGP operation of BGP-speaking routers. *(Pros)* Bitcoin nodes can obtain the up-to-date BGP path to the destination by making on-demand queries to the local BGP gateways. Minimum one query has to be made to obtain the BGP route to an IP address. For this, network operators' assistance is necessary; for example, cloud service providers, access ISPs, or campus networks can provide APIs to the Bitcoin nodes in their networks for requesting the AS path to a certain destination. Note that such APIs are already widely available in most legacy BGP routers; e.g., BGP look-up service in Looking Glass servers [33]. *(Cons)* Large networks, such as cloud service providers, often have multiple BGP exit gateways interfacing multiple different peering ASes across different regions. When querying the BGP path to a destination IP address, the returned path may differ depending on the specific BGP gateways to which the query is made. This is due to the well-known interaction between intra- and inter-domain routing protocols (e.g., hot-potato routing [55]). A practical

Table 2: A quick comparison between three notable BGP route estimation algorithms in the literature.

Algorithms by Authors	Input Data	Advantages
Mao et al. [37]	AS-level topology and business relationship	Lightweight, minimal dependencies
Qiu et al. [47]	(all above) and BGP feed data	More fine-grained and accurate estimation
Akhoondi et al. [1]	(all above) and estimated AS path lengths	Over-estimation of intermediate ASes

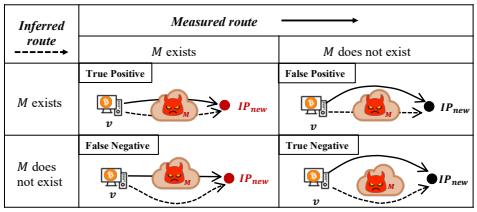


Figure 5: Confusion matrix for evaluating an inferred route from v to IP_{new}, given the potentially malicious AS M.

deployment of a control-plane look-up should take this issue into account.

(3) *Data-plane measurement:* A Bitcoin node directly measures the route towards a specific destination via probing tools, such as `traceroute`. *(Pros)* This approach requires minimal reliance on external data. Moreover, a Bitcoin node can obtain a fine-grained (i.e., IP-router level) paths that actual IP packets would likely travel. *(Cons)* This ***cannot*** be used in practice. The main problem is that a malicious Erebus AS can easily manipulate the `traceroute` measurements. The manipulation can be done by simply dropping the `traceroute` probe packets; worse, more careful manipulation of measured paths is also possible. Detection of such manipulation (e.g., anomaly detection in the longitudinal `traceroute` analysis) is extremely difficult because of the dynamic nature of route changes; see our anomaly detection of `traceroute` results in Appendix D. The bottom line is that due to the lack of authentication in the measured routing paths, detecting such spoofing is extremely challenging, making data-plane measurement an impractical option.

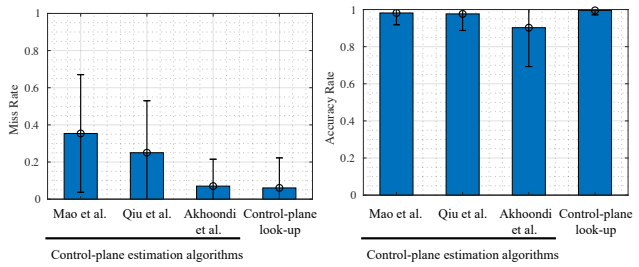
Evaluation setup. We, therefore, implement three state-of-the-art control-plane estimation schemes (i.e., *Mao et al.*'s algorithm [37], *Qiu et al.*'s algorithm [47], *Akhoondi et al.*'s algorithm [1]) and the control-plane look-up mechanism, and compare them particularly in terms of inference accuracy. We summarize the three control-plane estimation schemes in Table 2 and provide their brief descriptions as follows.

- *Mao et al.'s algorithm [37]* determines the inter-domain path between two ASes is the shortest AS path among all "valley-free" paths [22] between them based on the inferred business relationship [10]. We apply the following widely practiced BGP policies in order [23]: only valley-free paths are selected [22]; the shortest AS-path length route is preferred; and if multiple best paths exist, paths with smaller next-hop AS numbers are chosen.
- *Qiu et al.'s algorithm [47]* improves the AS path estimation for a prefix destination (instead of AS) by exploiting the known AS paths observed by globally-distributed BGP collectors. To implement this algorithm, we use the snapshot of 850 million AS paths to all IPv4 prefixes from 20 RIS collectors [44] and 25 Routeviews vantage points [49]. The collected AS paths are used to improve the simulation of the BGP propagation of these AS paths.
- *Akhoondi et al.'s algorithm [1]* does not output a single AS path between two ASes, unlike the other estimation algorithms, but over-estimates a set of ASes that the traffic likely traverses. The algorithm extracts all the segments of three consecutive ASes that appear in the collected AS paths from BGP collectors. Then it constructs all possible paths from the computed segments with the consideration of the given estimated length of the route between two ASes.[6] All unique ASes that appear in these paths are considered as the intermediate transit ASes between source and destination ASes.

For the control-plane look-up scheme, we use a PEERING client [50] to access the real-time inter-domain routes installed at all PEERING servers. While there exist other live BGP streaming frameworks that allow real-time access to the collected routes (e.g., BGPStream [11], RIS Live [43]), PEERING is the only publicly accessible platform that allows both control-plane look-up and data-plane route measurement to be performed on the same machine, to the best of our knowledge. To get the up-to-date BGP paths, we take the snapshots of the routing tables of all PEERING servers every hour.

To calculate the accuracy of the three control-plane estimation algorithms and the control-plane look-up mechanism, we test whether the inferred routes correctly identify a potentially malicious AS. If both the inferred and measured routes from a node v to the same IP address include a potentially malicious AS M, we consider the inferred route a True Positive. Figure 5 shows the confusion matrix that summarizes other evaluation outcomes. Similar to our evaluation in Section 3.2, we consider top-100 ASes as adversaries and our 59 measurement nodes as the victim Bitcoin nodes. Hence, we have nearly 6,000 pairs when evaluating the three control-plane estimation algorithms and 1,200 pairs when evaluating the control-plane look-up scheme (because the route look-up is only applicable to 12 PEERING nodes). With each attacker-victim pair, we use `TP`, `FP`, `FN`, and `TN` to represent the total number of

[6] The Lastor system in the original paper queries the estimated length between two arbitrary ASes from the iPlane platform [36], which unfortunately no longer operational at the time of this writing. Our implementation used the estimated length from the Qiu et al.'s algorithm instead.



(a) Miss Rate. (b) Accuracy Rate.

Figure 6: Average Miss Rate and Accuracy Rate with the standard deviation of the three control-plane estimation algorithms and the control-plane look-up scheme.

True Positive, False Positive, False Negative, and True Negative inferred routes, respectively, among routes to all IPv4 addresses. We compute the *Miss Rate* or the False Negative Rate (i.e., $\frac{\texttt{FN}}{\texttt{TP}+\texttt{FN}}$) to denote the probability of misidentifying an IP address as "not traversing AS M" when, in fact, the route to the IP address does include M. The Miss Rate is an important metric because it directly shows how often a victim node v would misidentify an IP address sent by the attacker AS M and connect to it. We also compute the accuracy (i.e., $\frac{\texttt{TP}+\texttt{TN}}{\texttt{TP}+\texttt{FP}+\texttt{FN}+\texttt{TN}}$) of all the route inference approaches.

Results. Figure 6a shows the average Miss Rate computed from all attacker-victim pairs for four different route inference mechanisms. Among all tested approaches, the control-plane look-up mechanism yields the lowest average Miss Rate of 0.06. This means that RAP implementing the control-plane look-up would miss, on average, only 6% of routes that include the malicious AS. The over-approximation algorithm by Akhoondi et al. [1] also results in a similar Miss Rate. The two other algorithms [37, 47] have the average Miss Rates of 0.35 and 0.25, respectively, making them unfit for the RAP implementation. Such high Miss Rates may seem incorrect to some, especially considering that these route estimation algorithms, in particular Mao et al.'s algorithm [37], have been so widely used in academic papers (e.g., [7, 19, 48, 58] to name a few) for more than a decade. This can be explained when we see the pretty high accuracy of most route estimation algorithms in Figure 6b. The accuracy of two widely used route inference models (see the two leftmost bars) is higher than 97%, explaining why they have been widely used in existing works. The algorithm by Akhoondi et al. [1] has much lower accuracy (e.g., 90%) because its primary purpose is to over-estimate the intermediary ASes, not to infer a single AS path accurately.

In a nutshell, most of the existing BGP route estimation algorithms are highly accurate in inferring the overall view of the BGP routes of today's Internet. When it comes to estimating an exclusion of a malicious AS M on a given path that does include M for RAP, control-plane look-up is the most suitable scheme with the lowest yet non-ideal Miss Rate of 6%.

5.2 How to *Exploit* Route-inference Errors

Both the adversary AS and the victim Bitcoin node with RAP can obtain the inferred routes and, as a result, an estimation of the attacker IPs set. However, only the adversary AS can obtain the ground-truth set of her IPs because the victim node cannot accurately obtain the measured routes; refer to Section 5.1 to see why direct route measurement by Bitcoin node is impractical. This offers a fundamental *asymmetric advantage* to the Erebus adversary compared to the victim Bitcoin node because the victim has no reliable way of learning a small subset of attacker IPs. Here, we discuss how the Erebus adversary can exploit this new advantage to enhance her attacks when RAP is deployed.

We clarify the key terminologies for two specific types of IP addresses that are useful for attacks:

- *Shadow IP*: An IP address whose *data-plane* route from the victim to itself includes a malicious AS. The attacker AS can utilize shadow IPs to create peering connections that will be under its control. We use the same terminology from the Erebus attack [57].
- ***Hidden-shadow IP***: A shadow IP address whose *inferred* victim-to-itself route does *not* include the malicious AS.

All the shadow IPs correspond to the union of the True Positive and False Negative cases in Figure 5 because the adversary AS M does exist on the data-plane paths towards the IP addresses. Among the shadow IPs, some are hidden-shadow IPs, and they correspond to the False Negative cases in Figure 5 as the inferred routes to hidden-shadow IPs do not include the adversary AS M.

Finding *hidden-shadow* IPs. An adversary AS can obtain the accurate shadow-IP set. She can simply send a victim Bitcoin node an IP packet with a spoofed source IP p, which triggers a response from the victim node (e.g., `ping`, `SYN`). If the adversary AS sees a corresponding response (e.g., `ICMP Echo Reply`, `SYN/ACK`) from the victim node, the IP p is confirmed shadow. To obtain the shadow-IP set, the adversary repeats the same process for all available IPv4 prefixes (about 800K) in the Internet. Note that the probe packets sent to the victim node have all different source IP addresses. To avoid suspicion, the adversary AS can spread out the probing over a longer period of time. Also, to reduce the number of probes to the victim node, the probing can be made to other public servers (e.g., `SSH`, `DNS`, `NTP`, `HTTP`, `HTTPS`) in the same subnet with the victim node. These public servers can be easily found with widely available scanning tools; e.g., Nmap [35], ZMap [16]. To obtain the hidden-shadow-IP set, the adversary can infer the routes to the enumerated shadow IPs by following the RAP defense's detailed algorithms, which are supposed to be publicly available, and remove the True Positive cases.

How many hidden-shadow IPs are found? Figure 7 shows the availability of hidden-shadow IPs and their distribution in 1,200 attacker-victim scenarios; i.e., adversaries are top 100

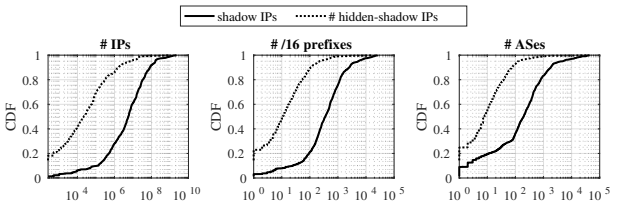


Figure 7: Cumulative distribution of shadow and hidden-shadow IPs in terms of their number of IPs, /16 prefixes, and ASes.

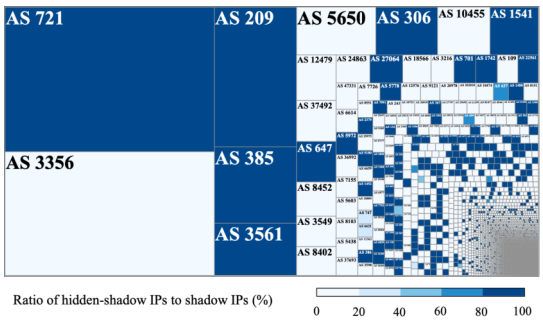


Figure 8: Number of shadow and hidden-shadow IPs in each AS from one experiment with a specific adversary (AS3356) and a victim (located in AS2637). The size of rectangles indicates the number of shadow IPs, and their darkness shows the ratio of hidden-shadow IPs to shadow IPs.

ASes, and victims are 12 PEERING servers.[7] We compare the shadow IPs and hidden-shadow IPs in terms of the CDF of IP address count in each type and the number of /16 prefixes and unique ASes hosting those IPs. Figure 7 shows that the number of hidden-shadow IPs is significantly smaller than the number of shadow IPs; for instance, there are only 24 thousand hidden-shadow IPs compared to 6 million shadow IPs at the median case, see the first plot. Overall, the hidden-shadow IPs are not plentiful; e.g., less than 15% of cases have more than a million hidden-shadow IPs, see the dotted line. Hidden-shadow IPs are also not well-diversified — in the vast majority of cases (e.g., 90%), they are hosted in only less than 120 distinct groups of /16 prefix and 80 unique ASes. This is much more concentrated than the shadow IPs, which are easily distributed in a few thousands of prefix groups and ASes in the majority of cases.

For better visualization of how hidden-shadow IPs are distributed, we plot the details of hidden-shadow IP allocation and their relationship with shadow IPs in a single attack example, where the attacker is Level3 (AS3356) and the victim node locates at Georgia Tech (AS2637), as shown in Figure 8. The number of shadow IPs in each AS is proportional to the area of rectangles and the ratio of hidden-shadow IPs to shadow IPs in that AS (up to 100%) is indicated by the darkness of the rectangles. In this example, shadow IPs are distributed in several thousands of ASes; yet, they tend to concentrate at only a handful of them — more than half of them belong to less than 10 ASes. Interestingly, Figure 8 also shows that in the majority of ASes, either all shadow IPs are also hidden-shadow IPs (i.e., RAP misidentifies all shadow IPs in this AS) or none of them is (i.e., RAP correctly identifies all shadow IPs).

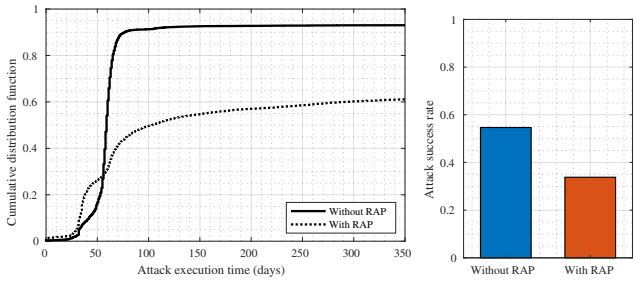
Exploiting *hidden-shadow* IPs to undermine RAP. When the victim implements RAP, the Erebus adversary can adaptively prioritize inserting the hidden-shadow IPs to the victim depending on the RAP defenses' publicly available configuration. For example, if the victim allows some of its estimated connections to share a common AS (e.g., $\tau = 5$), the attacker can select some regular, non-hidden-shadow IPs along with hidden-shadow IPs so that there are more attack IPs and they also become more diversified. When a low threshold τ is selected, the adversary may exclusively select and use hidden-shadow IPs to attack the victim. Since hidden-shadow IPs are quite limited in the majority of cases, there might be insufficient distinct IPs to poison the victim with a desirable attack rate (e.g., 2 IP/s). When this happens, the adversary repeatedly advertises the same hidden-shadow IP address from multiple source IP addresses to increase the appearance of a hidden-shadow IP in the victim's peer database.

5.3 How (in)effective is RAP?

We use the same evaluation framework described in Section 3.2, which includes about 6,000 attack scenarios, to evaluate the realistic impacts of RAP defense against the Erebus attacker who can exploit the hidden-shadow IPs. For the incorporation of RAP in the emulator, we choose the control-plane look-up mechanism because it achieves the lowest Miss Rate of only 6%. When the control-plane route look-up is not available, we synthesize the control-plane look-up results by making some randomly selected prefixes containing hidden-shadow IPs exclusively, adding a Miss Rate of 6% to the data-plane routes. We configure RAP to have the median threshold of $\tau = 5$ (i.e., the victim allows an AS to appear in at most five connections) in this evaluation and defer the detailed evaluation of other τ thresholds to Section 6.2.

Contrary to the common belief, the RAP defense does *not* demonstrate excessively powerful effectiveness. Figure 9 shows that when the victims do not implement RAP, the attacker hijacks all of their connections within 350 days in 93% of attack instances, and when RAP is included, 60% of instances are still successful, which is 33 percentage points of reduction. The RAP defense generally extends the required attack duration, yet, it is not significant. For example, to isolate 50% of the victims, the attackers need about 60 days when RAP is not deployed and no more than 100 days when the victims implement RAP. Moreover, considering the victim nodes' lifespan, the attack success rate when the victims implement RAP is still 33%, which is only 20 percentage

[7] We choose this evaluation set because the control-plane route look-up scheme has the highest accuracy, see Section 5.1.



(a) CDF of required attack time. (b) Attack success rate.

Figure 9: Cumulative distribution of the required execution time and the attack success rate when Bitcoin client implements RAP.

points lower than the baseline scenario, see Figure 9b. These results demonstrate that the RAP defense's robustness turns out to be insignificant when the Erebus attacker exploits the hidden-shadow IPs, unfortunately.

To better understand the effectiveness of the Erebus attacks with hidden-shadow IPs, we investigate how hidden-shadow IPs get selected to be the peers of the victim. Figure 10 shows how hidden-shadow IPs gradually occupy the `new` and `tried` tables in three scenarios, in which the adversary utilizes a small, medium, and large set of 10 thousand, 2.5 million, and 130 million hidden-shadow IPs, respectively. Note that we show only the ratio of legitimate reachable IPs and hidden-shadow IPs in the two tables in Figure 10 and exclude the non-hidden-shadow IPs because they are identified by the RAP defense and can occupy up to $\tau = 5$ connections, making the rest of the connections contested by legitimate reachable IPs and the hidden-shadow IPs. Figure 10 shows that in all three cases, the hidden-shadow IPs eventually dominate the legitimate ones. With the medium and large set of hidden-shadow IPs, the adversary can occupy the vast majority (e.g., $> 95\%$) of `new` table slots as well as the large part of the IPs (e.g., 70–80%) in the `tried` table, see Figure 10b and 10c. With these highly dominated ratios, it is easily understandable that the attacks can be successful within a few weeks. Figure 10a describes an interesting attack instance in which only 10 thousand repeatedly-advertised hidden-shadow IPs can dominate the legitimate IPs in the `new` table and occupy a non-negligible ratio in the `tried` table. This demonstrates that even a small amount of errors (e.g., 10 thousand misidentified IPs) can circumvent the RAP defenses.

5.4 RAP in Bitcoin vs. RAP in Tor

The RAP approach turns out to be a no silver bullet for the Erebus attack in Bitcoin, particularly because the infrequent route inference errors still allow the Erebus adversaries to find and use tens or hundreds of thousands of Sybil identities for eclipse attacks. We investigate whether the same weakness of the RAP approaches also seriously undermines the defense efficacy of previous RAP-based defenses in similar Tor attacks [1, 7, 45, 53], and find that the same weakness (despite its existence) cannot be exploited in Tor. The biggest reason is that, unlike the Bitcoin P2P network, Tor is not a fully permissionless system. Tor is only partially permissionless in the sense that anyone can volunteer to run Tor relays but new relays must go through some rigorous bandwidth review processes by the Tor infrastructure before they join the Tor P2P network [56]. Therefore, even if a network attacker in Tor finds large numbers of hidden-shadow IPs, she cannot use them as her Sybil identities. In fact, the limited effect of route-inference errors in RAP defenses in Tor has been studied in a recent work [32], and our work on RAP in Bitcoin shows a striking contrast.

Table 3: Attack success rates when victims implement combinations of two tweaks. Green indicates that the combination is better than both individual tweaks while yellow indicates that the combination is better than only one of them.

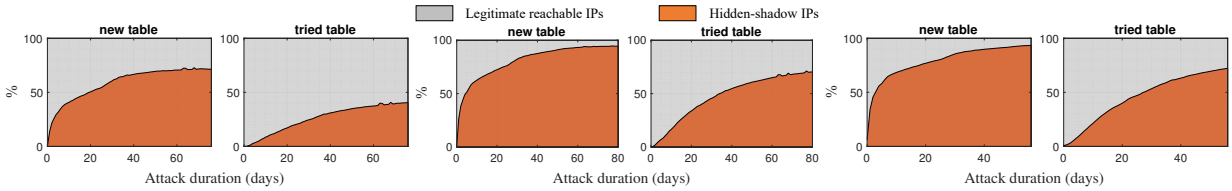
Baseline: 0.547	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
T1	**0.536**	0.422	0.410	0.403	0.390	0.591
T2		**0.386**	0.409	0.311	0.309	0.427
T3			**0.414**	0.301	0.291	0.449
T4				**0.406**	0.247	0.408
T5					**0.402**	0.387
T6						**0.609**

6 Practical Integrated Countermeasures

In previous sections, we discuss the limitations of several simple protocol patches and the more complex RAP approaches, and our empirical studies show that none of them sufficiently mitigate the Erebus attacks alone. The next seemingly obvious step is to compose some of these available countermeasures hoping that their overall effectiveness will be sufficient for handling the attacks in most practical scenarios. However, such an integrated countermeasure is non-trivial to design for two reasons: (1) the defense performance of available countermeasures seems non-monotonic; and (2) there is no one-size-fits-all RAP configuration in practice. This section makes several practical suggestions to Bitcoin, including finding a cost-effective combination of countermeasures (§6.1), and the location-specific optimal configuration for RAP operation (§6.2).

6.1 Balancing the Efficacy and Costs of Simple Countermeasures

Erebus countermeasures' efficacy is measured in the required Erebus attack execution time, which can be translated into the attack success rate in conjunction with the Bitcoin node churn rate; see Section 3.2 for details. The cost of countermeasures is less obvious to measure as it involves various forms of costs incurred in different solutions; yet, one clear rule of thumb is that the more countermeasures are activated, the more costly the overall integrated countermeasure.



(a) Small set: 10K hidden-shadow IPs (b) Medium set: 2.5M hidden-shadow IPs (c) Large set: 130M hidden-shadow IPs

Figure 10: Ratios of hidden-shadow IPs (orange) and legitimate reachable IPs (gray) in `new` and `tried` tables when adversaries find different numbers of hidden-shadow IPs.

Balancing the efficacy and costs of the available simple solutions is, unfortunately, hard. The main reason is the *non-monotonicity* of the efficacy of these countermeasures. To be specific, activating more countermeasures does *not* necessarily guarantee higher defense performance in terms of attack success rate. Thus, in practice, we need to evaluate many combinations and select a set of countermeasures given the allowed countermeasure costs. For example, if only **T1**, **T3**, and **T5** are allowed to run in a Bitcoin node, the best combination of these three simple countermeasures should be found after exhaustive evaluations of all possible combinations.

This section shows the non-monotonicity of these simple countermeasures and presents an example of choosing the best set of countermeasures.

Pairwise evaluation of T1–T6. To investigate the impact of the tweaks on each other's effectiveness, we test the combination of any two tweaks using our evaluation framework (See Section 3.2) and show the attacks' success rates against clients implementing such combinations in Table 3. Overall, the tweaks **T4** and **T5** demonstrate stronger defense performance when combining with other tweaks than they do alone. For instance, combining **T4** and **T5** brings the attack success rate down to 24.7%, which is 30 percentage points lower than the baseline (i.e., no countermeasures deployed). The tweaks **T1–T3** also show generally promising results when combining with others, except that a few combinations yield even worse defense performance than the individual tweaks (e.g., **(T1, T2)** versus **T2**), showing the non-monotonicity of these countermeasures in general. Also, the protocol tweak **T6** reduces the effectiveness of all other tweaks except **T5**.

An example. We consider a hypothetical Bitcoin node operator who is willing to try many simple countermeasures in this example. The operator, however, may learn that the performance of **T6** alone is even worse than the baseline and decide not to activate **T6**. The operator then evaluates all tweak combinations composed from the five other tweaks **T1–T5** and compares their defense effectiveness in Figure 11. In particular, when combining three tweaks **(T3, T4, T5)**, we see the lowest attack success rate of 20.9% (see the right-most blue bar) among all combinations of three tweaks. The orange bars show that adding **T1** or **T2** into **(T3, T4, T5)** only makes negligible improvements as the attack success rates

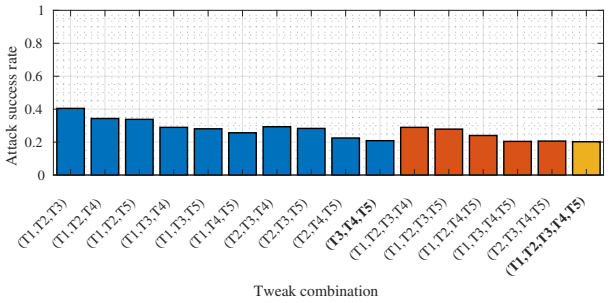
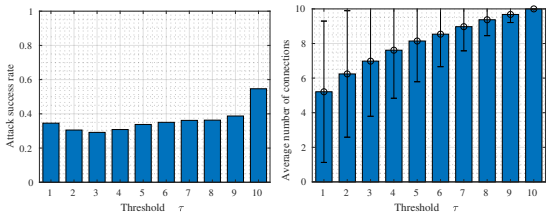


Figure 11: Attack success rates when Bitcoin nodes implement any combinations of tweaks **T1–T5**.



(a) Attack success rates. (b) Average numbers of established connections.

Figure 12: Attack success rates and the average number of established connections for different τ values.

are still 20.4% and 20.6%, respectively. Lastly, the right-most bar shows the combination of tweaks **(T1, T2, T3, T4, T5)** with a success rate of 20.2%, which is only minutely better than **(T3, T4, T5)**. Understanding this small performance gain from adding **T1** and **T2**, the operator may find the best operating point with the three tweaks **(T3, T4, T5)**. Note that this decision is given only as an example, and Bitcoin node operators with varying degrees of willingness to allow the simple tweaks may find other combinations more appropriate.

6.2 Location-based Customization of RAP

When operating the RAP defense (§4) in practice, a Bitcoin node operator should decide how strictly the RAP policy should be enforced. The threshold τ ($1 \leq \tau \leq 10$) is used to control this: the low value of τ strictly enforces the peer

connections to share a small number (or zero if $\tau = 1$) of common intermediate ASes, and a high value of τ allows many peer connections to share the same AS on their paths. It may seem straightforward to choose τ, as a more effective defense (e.g., a lower attack success rate) is expected with a lower τ value. Here, we show the opposite — choosing the proper threshold is non-trivial.

Effect of different τ values. To understand how different τ values affect the RAP effectiveness, we extend our experiment in Section 5.2 to evaluate RAP with all values of τ (i.e., $1 \leq \tau \leq 10$) against the Erebus attacks. We present the rate of successful attacks in almost 6,000 scenarios in Figure 12a. It shows that the overall attack success rates do not change significantly across different values of τ, and the rates even increase as τ decreases in the range $1 \leq \tau \leq 3$. This may look counterintuitive at first; however, it can be explained by Figure 12b in which we present the average number of established outgoing connections of the target nodes in our evaluations. It shows that our victim nodes have to avoid choosing many peers aggressively and may not have full connectivity to ten other nodes when the RAP is too strictly enforced (i.e., τ is set to a low value). For example, with $\tau = 1$, the victim nodes in our experiments can make only five connections on average. When the number of outgoing connections decreases, it becomes easier to hijack all of them!

Bitcoin node's location and τ. The results in Figure 12a show that strict enforcement of RAP (e.g., $\tau = 1$) yields a suboptimal defense performance when measuring the average performance *across all* 59 target Bitcoin nodes in our evaluation. We, however, conjecture that the defense performance of RAP may highly vary depending on the route diversity of a *specific* Bitcoin node, and thus the choice of τ should also consider the location of the node on the Internet topology. The rationale behind our conjecture is that some nodes in a well-connected network may be able to establish most of the ten outgoing connections even with a low τ value.

To see how different locations of Bitcoin nodes affect the choice of τ, we pick three examples in which the victims are located at vastly different topological locations on the Internet (i.e., two at cloud providers, one at university network) in Figure 13. We measure the attack success rate and the number of established connections for three specific Bitcoin nodes against the Erebus attacks from the top-100 large ASes. The τ values that yield the lowest attack success rates are $\tau = 1$, $\tau = 5$, and $\tau = 9$ for the three Bitcoin nodes, respectively. These best τ values coincide with the smallest τ values that make the victims fully connected with all ten outgoing connections in most cases. These results confirm that the choice of τ *significantly depends* on *where* on the AS topology individual Bitcoin nodes are located.

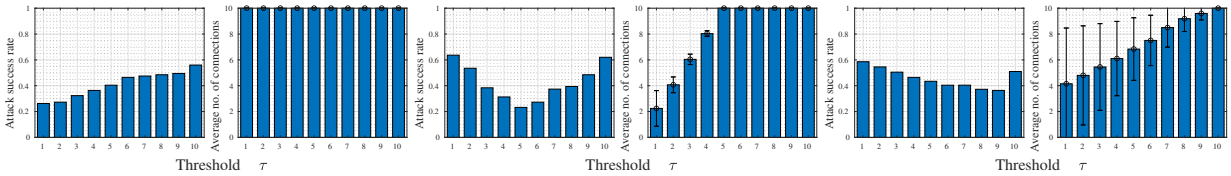
Finding the optimal τ. From the above experiments, we learn that it is desirable to choose a minimum possible τ value that ensures all ten outgoing connections are established. For an easier expression of this aspect, we define the desired lower-bound for available peer IPs, κ, set by each Bitcoin node operator. It ensures that there are at least κ IPs (among all reachable IPs in its database) that have not been marked as unavailable by RAP before any connection establishment.

Algorithm 1 Find the optimal threshold τ for Bitcoin node v.

Require: κ: the desired lower-bound for available IPs.
$G_1, G_2, ..., G_n$: groups of IPs having the same first-hop AS on their paths from v ($|G_1| \geq |G_2| \geq \cdots \geq |G_n| > 0$).
Ensure: $\tau_{optimal}$: the optimal threshold.

```
1:  procedure FindOptimalThreshold
2:      τ_optimal ← 10
3:      for τ ← 1 to 9 do                          ▷ try smaller thresholds first.
4:          A ← Sum(|G_1|, ··· , |G_n|)            ▷ number of available IPs.
5:          idx ← 1                                ▷ start with the biggest group G_1.
6:          cnt ← 0                                ▷ count IPs from the same group.
7:          for i ← 1···9 do                       ▷ try to select 9 peers.
8:              Peer_i ← G_idx.Pop()               ▷ select from G_idx.
9:              cnt ← cnt + 1
10:             A ← A − 1
11:             if cnt ≥ τ ∨ |G_idx| = 0 then      ▷ done with G_idx.
12:                 A ← A − |G_idx|
13:                 idx ← idx + 1                  ▷ move to the next group.
14:                 cnt ← 0
15:             end if
16:         end for
17:         if A ≥ κ then                          ▷ enough IPs for the 10^th connection.
18:             τ_optimal ← τ
19:             break
20:         end if
21:     end for
22:     return τ_optimal
23: end procedure
```

Algorithm 1 outlines an efficient computation for selecting an optimal τ value for RAP, given the topology of a Bitcoin node and its operator's desired κ value. At a high level, we test all τ values and consider the lowest threshold that satisfies κ as optimal. A threshold τ is said to satisfy κ if the IP availability for the last connection in the worst connectivity scenario (i.e., the lowest IP availability possible) is still sufficient. We greedily construct the worst scenario of connectivity based on two following intuitions. First, we can group all available IPs based on the first-hop AS on the paths from the victim node to them, whereas at most τ IPs can be chosen from each group during the connection establishments. When no first-hop AS appears in more than τ connections, it also implies that no other AS does because in all paths from the Bitcoin node, the occurrences of an AS is no more than the occurrences of the first-hop AS on the paths to that AS. Let us call $G_1, G_2, ..., G_n$ groups of IPs having the same first-hop ASes from the Bitcoin node to them. Second, to minimize the IP availability for the last connection, we should prefer the larger groups when establishing the other connections because all but τ IPs from those groups will become unavailable. Particularly, in Algorithm 1, we start with setting $\tau = 1$ for RAP (Line 3) and establish nine connections while preferring the peers from bigger IP groups (Line 7–16). We then check if the availability of IPs for the tenth connection satisfies κ and if it is, the current τ value is optimal; see Line 17–20. Otherwise, we repeat the above process with a higher τ value until the constraint regarding κ is satisfied.



(a) Victim location: London (AS16509) (b) Victim location: Bangalore (AS14061) (c) Victim location: Atlanta (AS47065)

Figure 13: Attack success rates and the average number of established connections of three victims.

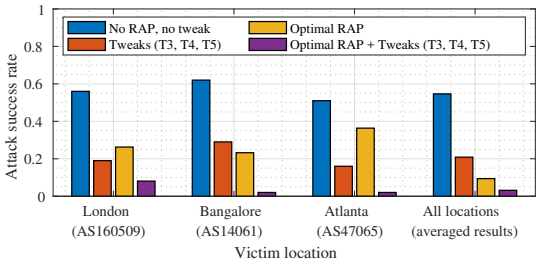


Figure 14: Attack success rate when three victims implement RAP combined with some protocol tweaks.

Combing RAP and tweaks. Here, we attempt to combine the optimal RAP defenses with some effective protocol tweak combinations discussed in previous sections to show their integrated defense performance. Following the example above, we implement the combination of the tweaks (**T3, T4, T5**) along with RAP with the optimal τ values[8] in all victim clients. We show the success rates of the attacks against them and highlight three victim examples (i.e., located in London, Bangalore, and Atlanta) in Figure 14. Figure 14 shows that combining RAP and tweaks makes the most effective countermeasure against the Erebus attacks as the attack success rates drop to 2–8% in three examples and only 3% on average.

6.3 Responsible Disclosure

We have disclosed our findings to the Bitcoin Core security team in late December 2020. The Bitcoin Core developers acknowledged the effectiveness of the RAP approach and are seeking extensive discussions before its deployment. Regarding the simple protocol tweaks, we learn that the team has been implementing a slightly different IP-to-ASN mapping for the tweak **T1**, see Appendix C for more details. Also, the tweaks **T2** and **T3** are being actively implemented while **T4** and **T5** are in consideration as of this writing. We will keep updating the status of these countermeasures on our public project webpage at https://erebus-attack-countermeasures.github.io/.

[8] When combining RAP and tweak **T3**, the optimal τ threshold for RAP should be adjusted accordingly to the increased number of connections.

7 Related Work

7.1 Security Research in Blockchain Networks

Security breaches in the P2P layer of blockchain networks often cause significant damage to the entire blockchain systems as the P2P layer is the fundamental underlying network of the consensus and transaction layers.

Eclipse attacks and defenses. In recent years, several attacks have shown that eclipsing the P2P networks of blockchain systems is possible [4, 8, 27, 28, 38, 57]. Earlier eclipse attacks [27] utilize small-size botnets and exploit specific vulnerabilities of the Bitcoin client software to partition one or more nodes from the P2P networks. Similar attacks that exploit the implementation bugs of Ethereum have also been presented [28, 38]. Most of these vulnerabilities have been quickly fixed by Bitcoin and Ethereum communities, rendering these attacks ineffective. More recent studies show that a malicious AS can control all the connections of a targeted Bitcoin node via launching a BGP hijacking attack [4]. As a solution to this Bitcoin hijacking attack, a new Bitcoin relay system, called SABRE [3], that is designed to be robust against BGP hijacking attacks. There also exists an eclipse attack that specifically targets Bitcoin nodes connecting via Tor bridges by exploiting the anti-DoS mechanism in Bitcoin [8].

Relays and Bitcoin security. Although Bitcoin is designed as a fully decentralized P2P network, a special type of peer nodes, called relays, have been proposed for performance and security purposes. Fast relay networks, such as Falcon [18] and FIBRE [20], have been used to speed up the block data propagation between a closed group members. SABRE [3] is a special relay that guarantees BGP-hijacking-free peering. One may be tempted to rely on such existing relays or even introduce more relays to diversify his/her Bitcoin nodes' connectivity in the hope that it will mitigate the Erebus attacks. However, it is a far-from-ideal approach to handling the Erebus attacks. First, there still exist non-negligible chances that some malicious transit ASes are on the paths to the relays. Second, perhaps more importantly, the reliance on a small number of relays for P2P operations makes the relay infrastructure effectively a *centralized regulating authority*. Relays may be used as a temporary measure when no good countermeasures to Erebus exist but they cannot be a long-term

solution. In this paper, we focus on the solutions that do not hurt the openness and decentralization of Bitcoin.

7.2 Routing Awareness in Tor

Our work is the first to thoroughly evaluate the ideas of routing awareness in blockchain systems. In fact, the idea of RAP has been already investigated several times in Tor [1, 7, 17, 19, 32, 45, 53]. These Tor systems employ routing awareness by using inter-domain route inference algorithms (e.g., Mao et al.'s algorithm [37]) to estimate a common, suspicious AS that appears on both connections from a client to a Tor entry node and from a Tor exit node to a destination, similar to how we implement RAP in Bitcoin. Juen et al. [32] also compare the inferred AS paths of the Tor connections with the data-plane paths and report an overall 80% difference between them. These routing-aware mechanisms developed for Tor cannot be directly used for Bitcoin because the semi-permissionless nature of Tor network is fundamentally different from that of Bitcoin, see more detailed discussion in Section 5.4.

8 Conclusion

Perhaps, we may have been building a house of cards when we keep inventing new blockchain consensus ideas while relying on the P2P networks that are easy to eclipse. We attempt to address one recent eclipse attack that exploits a powerful network-Sybil capability, with the practicality of countermeasures as the top priority. Our critical evaluation shows that one highly promising countermeasure, called routing-aware peering (RAP), yields disappointing defense performance due to its weakness. Yet, we confirm that Bitcoin can be protected from most Erebus attacks when these available countermeasures are carefully optimized and customized for each node. We believe that our work helps us take a step towards highly reliable blockchain P2P networking protocols.

Acknowledgments

We thank the anonymous reviewers of this paper and our shepherd Yixin Sun for their helpful feedback. We also thank Inho Choi for the useful comments on early versions of this paper and help in `traceroute` experiments. This work uses measurements from the PEERING testbed, which cannot be done without support from Italo Cunha and other team members. We also thank Gleb Naumenko and other Bitcoin Core developers for the discussion on RAP and protocol tweaks presented in this paper. This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01343, Regional strategic industry convergence security core talent training business) and the CRYSTAL Centre at National University of Singapore.

References

[1] Masoud Akhoondi, Curtis Yu, and Harsha V Madhyastha. LASTor: A low-latency AS-aware Tor client. In *Proc. IEEE S&P*, 2012.

[2] Ehab Al-Shaer, Qi Duan, and Jafar Haadi Jafarian. Random host mutation for moving target defense. In *Proc. SecureComm*, 2012.

[3] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. SABRE: Protecting Bitcoin against Routing Attacks. In *Proc. NDSS*, 2019.

[4] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing attacks on cryptocurrencies. In *Proc. IEEE S&P*, 2017.

[5] asmap-rs. https://github.com/rrybarczyk/asmap-rs, 2020.

[6] Erebus Attack. Bitcoin emulator, 2020. https://github.com/Erebus-Attack/Bitcoin-Emulator.

[7] Armon Barton and Matthew Wright. DeNASA: Destination-Naive AS-Awareness in Anonymous Communications. In *Proc. PETS*, 2016.

[8] Alex Biryukov and Ivan Pustogarov. Bitcoin over Tor isn't a good idea. In *Proc. IEEE S&P*, 2015.

[9] CAIDA. AS Rank: A ranking of the largest Autonomous Systems (AS) in the Internet, 2020. https://asrank.caida.org/.

[10] CAIDA. AS Relationships, 2020. http://www.caida.org/data/as-relationships/.

[11] CAIDA. BGPStream, 2020. https://bgpstream.caida.org/.

[12] CAIDA. Routeviews Prefix to AS mappings Dataset (pfx2as) for IPv4 and IPv6, 2020. https://www.caida.org/data/routing/routeviews-prefix2as.xml.

[13] Bitcoin Core. Bitcoin Core 0.21.0, 2021. https://bitcoincore.org/en/releases/0.21.0/.

[14] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin P2P network. In *Proc. FC*, 2014.

[15] John R Douceur. The Sybil attack. In *Springer IPTPS*, 2002.

[16] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *Proc. USENIX Security*, 2013.

[17] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In *Proc. ACM CCS*, 2009.

[18] Falcon. A Fast Bitcoin Backbone, 2016. https://www.falcon-net.org/.

[19] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In *Proc. ACM WPES*, 2004.

[20] FIBRE. Fast Internet Bitcoin Relay Engine, 2020. http://bitcoinfibre.org/.

[21] William Foxley. Latest Bitcoin Core Code Release Protects Against Nation-State Attacks. *CoinDesk*, 2020.

[22] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM TON*, 2001.

[23] Phillipa Gill, Michael Schapira, and Sharon Goldberg. A survey of interdomain routing policies. *ACM SIGCOMM CCR*, 2013.

[24] Jivika Govil, Jivesh Govil, Navkeerat Kaur, and Harkeerat Kaur. An examination of IPv4 and IPv6 networks: Constraints and various transition mechanisms. In *Proc. IEEE Southeast-Con*, 2008.

[25] Ethan Heilman. Added test-before-evict discipline in Addrman, feeler connections, 2015. https://github.com/bitcoin/bitcoin/pull/6355.

[26] Ethan Heilman. net: Feeler connections to increase online addrs in the tried table, 2016. https://github.com/bitcoin/bitcoin/pull/8282.

[27] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *Proc. USENIX Security*, 2015.

[28] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing Ethereum Peers with False Friends. In *Proc. IEEE EuroS&PW*, 2019.

[29] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. Churn in the Bitcoin Network: Characterization and impact. In *Proc. ICBC*, 2019.

[30] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proc. HotSDN*, 2012.

[31] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011.

[32] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar. Defending tor from network adversaries: A case study of network path prediction. In *Proc. PETS*, 2015.

[33] Thomas Kernen. Public route server and looking glass site list, 2011. http://www.traceroute.org/.

[34] Matthew Luckie. Scamper: a scalable and extensible packet prober for active measurement of the internet. In *Proc. ACM IMC*, 2010.

[35] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[36] Harsha V Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An information plane for distributed services. In *Proc. OSDI*, 2006.

[37] Z Morley Mao, Lili Qiu, Jia Wang, and Yin Zhang. On AS-level path inference. In *ACM SIGMETRICS PER*, 2005.

[38] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network, 2018. https://eprint.iacr.org/2018/236.

[39] Steven J Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In *Proc. PETS*, 2007.

[40] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[41] Gleb Naumenko. p2p: supplying and using asmap to improve IP bucketing in addrman, 2020. https://github.com/bitcoin/bitcoin/pull/16702.

[42] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. Erlay: Efficient Transaction Relay for Bitcoin. In *Proc. ACM CCS*, 2019.

[43] RIPE NCC. RIS Live - RIPE Network Coordination Centre, 2020. https://ris-live.ripe.net/.

[44] RIPE NCC. RIS Raw Data, 2020. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data.

[45] Rishab Nithyanand, Oleksii Starov, Adva Zair, Phillipa Gill, and Michael Schapira. Measuring and mitigating AS-level adversaries against Tor. In *Proc. NDSS*, 2016.

[46] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services, 2020. https://www.planet-lab.org/.

[47] Jian Qiu and Lixin Gao. AS path inference by exploiting known AS paths. In *Proc. IEEE GLOBECOM*, 2005.

[48] Tongqing Qiu, Lusheng Ji, Dan Pei, Jia Wang, Jun (Jim) Xu, and Hitesh Ballani. Locating Prefix Hijackers using LOCK. In *Proc. USENIX Security*, 2009.

[49] Routeviews. University of Oregon Route Views Project, 2020. http://www.routeviews.org/routeviews/.

[50] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *Proc. ACM CoNEXT*, 2019.

[51] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. In *Proc. ACM SIGOPS European Workshop*, 2004.

[52] Hennadii Stepanov. Try to preserve outbound block-relay-only connections during restart, 2020. https://github.com/bitcoin/bitcoin/pull/17428.

[53] Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *Proc. IEEE S&P*, 2017.

[54] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing attacks on privacy in Tor. In *Proc. USENIX Security*, 2015.

[55] Renata Teixeira, Aman Shaikh, Tim Griffin, and Jennifer Rexford. Dynamics of hot-potato routing in IP networks. In *Proc. of SIGMETRICS*, 2004.

[56] Tor. The lifecycle of a new relay. https://blog.torproject.org/lifecycle-new-relay, 2013.

[57] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In *Proc. IEEE S&P*, 2020.

[58] Jian Wu, Ying Zhang, Z Morley Mao, and Kang G Shin. Internet routing resilience to failures: analysis and implications. In *Proc. ACM CoNext*, 2007.

[59] Addy Yeow. Global Bitcoin nodes distribution, 2020. https://bitnodes.io/.

A A Large-scale Data-plane Route Measurement

Figure 15: A map of geographic locations of our 21 cloud instances (red pins), 26 PlanetLab nodes (blue pins), and 12 PEERING servers (black pins).

We perform a large-scale measurement to record the data-plane routes from 59 distributed nodes across the world to all available IPv4 prefixes. Particularly, in December 2019, we send out in total 47.2 million `traceroute` probes from 21 instances hosted at different regions of five popular cloud providers (i.e., Amazon, OVH, DigitalOcean, Hetzner, and Alibaba), 26 PlanetLab nodes [46], and 12 PEERING servers [50]. We visualize the geographical distributions of our measurement nodes in Figure 15. We note that none of these nodes are located at top-100 ASes, hence there is no overlapping with the list of attackers that we consider in most of our experiments in this paper. The destinations of the `traceroute` probes are approximate 800 thousand IP addresses that are randomly selected from all IPv4 prefixes in the Internet. We assume all IPs in the same prefix would have the same route as the randomly selected IP. We perform the measurements in parallel using the state-of-the-art tool `scamper` [34] at the rate of 400 packets per second where each set of measurements from one node to all destinations is finished in less than 20 hours. Then, we use the Routeviews Prefix-to-AS mapping [12] to convert the `traceroute` results into AS-level paths. Finally, we remove all unreachable hosts or hosts that do not belong to any mapping AS in all measured paths.

B IP churn rate of Bitcoin nodes

For a more realistic evaluation of the Erebus attacks, we study the IP churn rate of the actual Bitcoin nodes by measuring their online duration before they leave the system. We retrieve two years' worth of data (from January 1, 2018, to December 31, 2019) provided by Bitnodes — an online service that periodically takes the snapshots of all reachable Bitcoin addresses [59]. We consider an IP address that appears in two consecutive Bitnodes snapshots is online between two timestamps when the snapshots were taken. Following this, we compute the total online duration (in days) of about 340 thousand distinct Bitcoin addresses observed in this two-year period and plot the distribution in Figure 16. Figure 16 shows

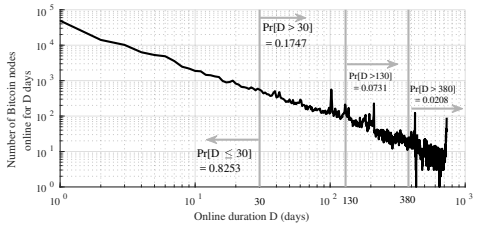


Figure 16: The online duration distribution (in days) of 340 thousand reachable Bitcoin nodes observed by Bitnodes [59] in two years from January 1, 2018 to December 31, 2019. Pr[·] indicates the empirical probability distribution of the online duration.

that the Bitcoin network has a high churn rate. The vast majority (e.g., 82.53%) of the Bitcoin nodes are fairly short-lived — they become unreachable within 30 days, which is well-aligned with an existing one-month measurement done by Donet et al. [14]. On the other hand, only 7.3% of the nodes were online for more than 130 days and 2% of nodes (i.e., about 6.8 thousand IPs) were online for over 2 years.

The IP churn rate distribution is crucial when it comes to the evaluation of the Erebus attacks. When evaluating whether the attack is successful against a specific victim, we calculate the required attack execution time and consider a random online duration from the distribution to be the lifespan of the victim node. Note that short-lived nodes (i.e., nodes that are online for less than 30 days) are excluded from the consideration because the usual targets of Erebus attacks are long-lived and highly influential Bitcoin nodes, such as mining pool gateways [57].

C Implementations of IP-to-AS Mapping

The basic principle of the tweak **T1** (i.e., ASN-based grouping) is an IP-to-ASN mapping that maps any IP address to the AS number of the AS representing it. Given a set of attacker IPs, a mapping that groups them in a fewer number of groups is generally preferred because they will likely occupy fewer slots in the two tables, thus decreasing the attack success rate. In this paper, we take a simple approach that maps an IP address to the ASN of its actual owner, i.e., an *IP-to-owner-AS* mapping. During our disclosure with Bitcoin Core developers (see Section 6.3), we learn that they are implementing an uncommon, more sophisticated *IP-to-bottleneck-AS* mapping, which was still being discussed at the time we evaluated the tweak **T1** [5]. In this section, we describe this IP-to-bottleneck-AS implementation and show that it is only marginally different from our mapping implementation in terms of effectiveness.

Particularly, the IP-to-bottleneck-AS mapping maps an IP address to a so-called *bottleneck AS*, which is the first common AS that appears on all AS-paths collected by multiple

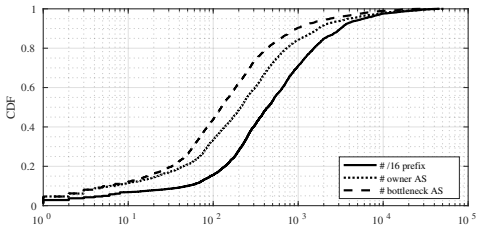


Figure 17: Cumulative distribution of attacker IPs in terms of their number of /16 prefixes, owner ASes, and bottleneck ASes.

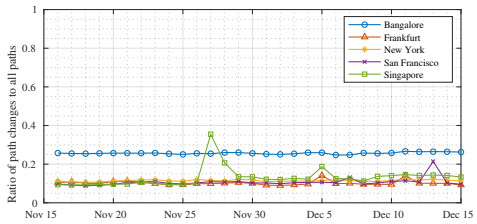


Figure 18: The ratio of AS-level paths that have changed compared to the previous day. The measurements are done from five DigitalOcean regions in a one-month period (from November 15, 2019, to December 15, 2019).

BGP collectors (e.g., RIPE RIS [44]) that destined to the IP address. Similarly, a Bitcoin operator can individually calculate the mapping with AS-paths retrieved from local routing table dumps. Intuitively, if attacker IPs are distributed in multiple single-homed ASes, this implementation would make a smaller number of groups than the IP-to-owner-AS mapping because the IPs tend to share common upstream ASes. On the other hand, it can be worse if a majority of IPs are from the same multi-homed AS and mapped to multiple bottleneck ASes.

To highlight the differences between three mapping options (i.e., the original /16 prefix, IP-to-owner-AS, and IP-to-bottleneck-AS), we show the distributions of attacker IPs in 5900 scenarios (see Section 3.2) in term of number of /16 prefixes, number of owner ASes, and number of bottleneck ASes in Figure 17. It shows that using both IP-to-AS mappings makes smaller numbers of groups than using the /16 prefix mapping. Also, the IP-to-bottleneck-AS mapping is shown to be insignificantly better than our IP-to-owner-AS implementation, e.g., the number of cases having 100 or fewer groups is increased by only ten percentage points.

D (In)stability of Data-plane Routes

The third option of the inference logic is the data-plane measurement scheme — a Bitcoin node measures the routes via sending `traceroute` probes to its peers. The main problem of this approach, which makes it unusable, is that a malicious AS, which is already on the paths of the `traceroute` probes, can easily manipulate the measurement results so that the paths look benign (e.g., the results do not include the malicious AS). For instance, the malicious AS can respond to the probe with another AS's IP address, or simply drop the probing packet.

One may argue that a victim Bitcoin node can still detect such manipulation by looking for anomalies of the `traceroute` paths based on its longitudinal history. For example, if the AS-level paths obtained via `traceroute` probes to the same IP address suddenly change, it may indicate that the `traceroute` probes may have been manipulated by a malicious AS. Our analysis, however, shows that such detection is difficult in practice. The reason is that the AS-level paths are already quite unstable, even when there exist no such attacks, and thus it is hard to distinguish sudden AS-path changes due to malicious activities from the frequent benign AS-path changes.

Here, we investigate the natural instability of `traceroute` measurements at the AS-level. Our analysis begins with measuring the data-plane paths from five nodes hosted at different regions of the DigitalOcean network (i.e., Bangalore (India), Frankfurt (Germany), New York (US-East), San Francisco (US-West), and Singapore) to all 800 thousand available IPv4 prefixes in the Internet, see Appendix A. The same set of experiments is repeated every day for one month (from November 15, 2019, to December 15, 2019). In each day, we compute the number of AS paths that have changed in comparison with the paths to the same destinations one day earlier. Figure 18 shows the ratio of path changes to all paths in each day from nodes in five regions. The data-plane paths at AS-level appear to be quite stable in general — about 90% of the paths measured from all regions (except Bangalore) are unchanged within one day period. However, this also means that 10% of the measured AS-level paths (i.e., about 80K) change within one day period. The measurements at Bangalore show a more dynamic behavior of the changing paths, in which about one-fourth of the paths are different from what they were in the previous day.

This non-negligible amount of AS-level path changes makes it difficult to find a practical threshold for anomaly detection. Worse, a malicious AS, knowing that a portion of the paths from the victim would change frequently, can slowly manipulate the `traceroute` results within the threshold to circumvent the detection. In conclusion, detecting the manipulation in data-plane measurements is extremely challenging and thus the data-plane measurement scheme cannot be used in practice for RAP.