Diss. ETH No.27210

Routing Security of Cryptocurrencies

A thesis submitted to attain the degree of

Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by
Maria Apostolaki

born on 12.02.1992
citizen of Greece

accepted on the recommendation of
Prof. Dr. Laurent Vanbever, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Sharon Goldberg, co-examiner

2021

TIK-SCHRIFTENREIHE NR. 190

Maria Apostolaki

Routing Security of Cryptocurrencies

A dissertation submitted to ETH Zurich
for the degree of Doctor of Sciences

DISS. ETH NO. 27210

Prof. Dr. Laurent Vanbever, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Sharon Goldberg, co-examiner

Examination date: July 12 , 2021

To mom and dad

Abstract

Cryptocurrencies are digital money operated by a set of nodes. The oldest and most widely-used cryptocurrency today, namely Bitcoin, gained its popularity thanks to its security properties: its openness, immutability, and anonymity. Indeed, Bitcoin offers all users an *open* platform to perform *immutable* transactions while hiding their *real-world identity*. Instead of relying on a central authority, Bitcoin nodes build an extensive overlay network between them and use consensus to agree on a set of transactions that are recorded within Bitcoin's core data structure: the blockchain.

Bitcoin nodes communicate over the Internet infrastructure, which is composed of multiple networks called Autonomous Systems (ASes). In effect, any AS on the Internet forwarding path between two nodes can access the messages they exchange. The Bitcoin protocol does not specify how the Bitcoin peer-to-peer overlay network should be mapped to the Internet to maintain its security properties. As a result, Bitcoin's security properties are in practice at risk. In this thesis, we aim at shedding light on the interactions between the application and the network layer by answering two questions.

Attack surface: What could be the impact of an AS-level adversary on the security properties of cryptocurrencies and Bitcoin in particular?

Defense: How can we shield such systems from an AS-level adversary?

In response to the first question, we prove that a single AS is able to compromise Bitcoin's immutability, anonymity, and openness. In response to the second question, we show that we can protect cryptocurrencies from AS-level adversaries by leveraging Internet policies, state-of-the-art networking hardware and cross-layer awareness.

We start these efforts by analyzing the Bitcoin network from the routing perspective. Our findings contradict a core assumption about the Bitcoin network, namely its decentralization. From the application perspective, Bitcoin is indeed decentralized, as it is an open network of thousands of independent nodes that establish random connections. From the routing perspective, though, the Bitcoin network is highly centralized, as few ASes intercept a disproportionately large amount of Bitcoin traffic. Driven by this insight, we introduce a new attack

vector, namely *routing attacks*. We uncover and ethically performed in the wild three novel routing attacks against Bitcoin: the partition, the delay, and the perimeter attack. These attacks generalize to other cryptocurrencies such as Ethereum. Other than the attacker's goal, the three attacks differ in terms of their effectiveness and detectability. On the one hand, the partition attack is extremely powerful – even against the Bitcoin network as a whole – but it can be easily detected. On the other hand, the delay and the perimeter attacks are only effective against a targeted set of nodes, yet they are very hard to be detected.

To shield Bitcoin against the partition attack we design SABRE, a relay network that keeps the Bitcoin network connected even in the presence of an AS-level attacker. SABRE achieves this by leveraging Internet routing policies and emerging networking hardware. Notably, SABRE's design is general and can be used to protect *any blockchain system* from such attacks.

Finally, to shield Bitcoin from harder-to-detect attacks such as the delay and the perimeter attack, we suggest a set of cross-layer countermeasures. In particular, we revisit the design and deployment choices of the Bitcoin (and the Ethereum) system to account for the risk of an AS-level adversary.

Zusammenfassung

Cryptocurrencies sind digitales Geld, das von einer Reihe von Knoten betrieben wird. Die älteste und heute am weitesten verbreitete Kryptowährung - Bitcoin - gewann ihre Popularität dank ihrer Sicherheitseigenschaften: ihrer Offenheit, Unveränderlichkeit und Anonymität. Tatsächlich bietet Bitcoin allen Benutzern eine *offene* Plattform, um *unveränderliche* Transaktionen durchzuführen, während sie ihre *reale Identität* verbergen. Anstatt sich auf eine zentrale Instanz zu verlassen, bauen Bitcoin-Knoten ein umfangreiches Overlay-Netzwerk auf und verwenden ein Konsensprotokoll, um sich auf eine Reihe von Transaktionen zu einigen, die in der Kerndatenstruktur von Bitcoin, der Blockchain, aufgezeichnet werden.

Bitcoin-Knoten kommunizieren über die Internet-Infrastruktur, die aus mehreren Netzwerken besteht, die als autonome Systeme (AS) bezeichnet werden. Tatsächlich kann jedes AS auf dem Internet-Weiterleitungspfad zwischen zwei Knoten auf die Nachrichten zugreifen, welche die Knoten austauschen. Das Bitcoin-Protokoll legt nicht fest, wie das Bitcoin-Peer-to-Peer-Overlay-Netzwerk dem Internet zugeordnet werden soll, um seine Sicherheitseigenschaften beizubehalten. Dadurch sind die Sicherheitseigenschaften von Bitcoin in der Praxis gefährdet. In dieser Arbeit wollen wir die Wechselwirkungen zwischen der OSI Anwendungs- und der Netzwerkschicht durch die Beantwortung zweier Fragen beleuchten.

Angriffsoberfläche: Welche Auswirkungen könnte ein Gegner auf AS-Ebene auf die Sicherheitseigenschaften von Cryptocurrencies und insbesondere von Bitcoin haben?

Verteidigung: Wie können wir solche Systeme vor einem Gegner auf AS-Ebene schützen?

Als Antwort auf die erste Frage beweisen wir, dass ein einzelnes AS in der Lage ist, die Unveränderlichkeit, Anonymität und Offenheit von Bitcoin zu gefährden. Als Antwort auf die zweite Frage zeigen wir, dass wir Cryptocurrencies vor Gegnern auf AS-Ebene schützen können, indem wir Internetrichtlinien, modernste Netzwerkhardware und OSI-schichtübergreifendes Wissen nutzen.

Um die Macht von Gegnern auf AS-Ebene einzuschätzen, analysieren wir zunächst das Bitcoin-Netzwerk aus der Routing-Perspektive. Unsere Ergebnisse widersprechen einer Kernannahme über das Bitcoin-Netzwerk, nämlich seiner Dezentralisierung. Aus Anwendungssicht ist Bitcoin tatsächlich dezentralisiert, da es sich um ein offenes Netzwerk aus Tausenden von unabhängigen Knoten handelt, die zufällige Verbindungen herstellen. Aus der Routing-Perspektive ist das Bitcoin-Netzwerk jedoch stark zentralisiert, da nur wenige AS eine unverhältnismäßig grosse Menge an Bitcoin-Verkehr weiterleiten. Motiviert durch diese Erkenntnis, führen wir einen neuen Angriffsvektor ein, nämlich

Routing-Angriffe. Um deren Relevanz aufzuzeigen, decken wir drei neuartige Routing-Angriffe gegen Bitcoin auf und führen sie ethisch aus: den Partitions-, den Verzögerungs- und den Perimeterangriff. Diese Angriffe lassen sich auch auf andere Cryptocurrencies verallgemeinern, wie Ethereum. Neben dem Ziel des Angreifers unterscheiden sich die drei Angriffe in ihrer Effektivität und Erkennbarkeit. Der Partitionsangriff ist extrem mächtig – auch gegen das Bitcoin-Netzwerk als Ganzes – aber er lässt sich leicht erkennen. Im Gegensatz dazu sind der Verzögerungs- und der Perimeterangriff nur gegen eine gezielte Gruppe von Knoten wirksam, aber sie sind sehr schwer zu erkennen.

Um Bitcoin gegen den Partitionsangriff abzuschirmen, entwerfen wir SABRE, ein Relay-Netzwerk, welches das Bitcoin-Netzwerk auch in Anwesenheit eines Angreifers auf AS-Ebene verbunden hält. SABRE erreicht dies durch den Einsatz von Internet-Routing-Richtlinien und neuartiger Netzwerkhardware. Insbesondere ist das Design von SABRE allgemein gehalten und kann verwendet werden um *jedes Blockchain-System* vor solchen Angriffen zu schützen.

Um Bitcoin vor schwerer zu erkennenden Angriffen, wie dem Verzögerungs- und dem Perimeterangriff, zu schützen, schlagen wir eine Reihe von schichtübergreifenden Gegenmassnahmen vor. Insbesondere überprüfen wir die Design- und Bereitstellungsentscheidungen des Bitcoin- (und des Ethereum)-Systems, um dem Risiko eines Gegners auf AS-Ebene Rechnung zu tragen.

Introduction

Cryptocurrencies are a form of digital currency that has the potential to revolutionize our financial system with their many advantages. First, cryptocurrencies allow users word-wide to perform private and secure transactions despite transaction data being public. Second, cryptocurrencies open up financial options for people in countries who lack access to financial services or suffer from high inflation. Third, cryptocurrencies are decentralized, giving users complete control over their funds, meaning that no bank, corporation, central government, or compromised vendor can access these funds without direct permission.

Due to their attractive properties and open codebase, cryptocurrencies have received tremendous attention from the academic community. Academics have sofar studied various aspects of such currencies and in particular Bitcoin, revealing various attack vectors such as double spending [88], eclipsing [65], transaction malleability [50], attacks targeting mining [52, 89, 82, 51] or user privacy [34, 71].

Still, one attack vector has been overlooked: attacking Bitcoin via the Internet infrastructure using *routing attacks*. Bitcoin connections are routed over the Internet, which is comprised of multiple independent networks, namely Autonomous Systems (ASes). In effect, any AS on the forwarding path can eavesdrop, drop, modify, inject, or delay Bitcoin messages such as blocks or transactions. Notably, the forwarding path is devised by BGP: the defacto routing protocol in the Internet. Thus, the application (e.g., Bitcoin) cannot control which ASes will forward and access application traffic. In effect, locating the attacker and mitigating her attack are highly challenging, even after the attack has been detected.

Locating AS-level attackers is challenging as it requires inferring the exact forwarding paths taken by the Bitcoin traffic using measurements (e.g., traceroute) or routing data (BGP announcements), both of which can be forged [86]. Even ignoring locating challenges, mitigating AS-level attacks is hard as it is essentially a human-driven process consisting of filtering, routing around, or disconnecting the attacker. As an illustration, it took Google more than 2 hours to resolve rogue BGP announcements affecting Google Cloud traffic [10] in 2018. Other examples of routing attacks such as [95] (resp. [96]) took 9 (resp. 2) hours to resolve in November (resp. June) 2015.

The main reason why routing attacks have been overlooked is that cryptocurrency networks are only studied from the application-layer perspective, making implicit and unrealistic assumptions about the underlying network, i.e., the Internet. Such assumptions include that there is no AS-level adversary that intercepts enough connections to threaten the security of the system itself or of individual users.

Yet, two characteristics of the Internet's infrastructure make AS-level adversaries practical today. First, BGP is highly insecure, despite the multiple proposed fixes [97, 37, 45]. Thus, multiple individual ASes can manipulate global routing to gain access over the connections of almost all Bitcoin nodes. Second, due to the centralization of its traffic to the Internet core, some ASes naturally (i.e., according to BGP) intercept a significant fraction of all Bitcoin connections.

Beyond the Internet characteristics, though, one more factor makes AS-level adversaries particularly practical: Bitcoin's centralization. By analyzing the Bitcoin network from the routing perspective, we found that the distributions of its mining power and of traffic in the Internet are highly skewed. Particularly, our analysis revealed two unintuitive insights, which generalize to other cryptocurrencies, such as Ethereum:

I1 A few ASes host a large portion of the nodes and the mining power.
I2 A few ASes intercept a large portion of Bitcoin connections.

The first insight (***I1***) most likely stems from the way individuals deploy Bitcoin clients or mining pools. As an intuition, they often use cloud providers to benefit from massive economies of scale. The second insight (***I2***) stems from ***I1*** combined with the peer selection algorithm in Bitcoin and the Internet's centralization.

Driven by these two insights, we classify routing attacks into two categories: active and passive, based on the particular insight that the attacker exploits. *Active attackers* manipulate Internet routing to forcefully gain access over Bitcoin connections. Such attackers are particularly effective thanks to the concentration of Bitcoin nodes and mining power to a few ASes (***I1***). *Passive attackers* operate on the traffic they naturally intercept. In effect, ASes intercepting a large portion of Bitcoin connections are particularly powerful (***I2***). To prove the menace of both categories, we uncovered and ethically performed in the wild three novel routing attacks: the partition, the delay, and the perimeter attack. The former

attack is active, while the latter two attacks are passive. While we evaluate the attacks for the case of Bitcoin, they all generalize to other cryptocurrencies.

The partition attack is an active attack that prevents Bitcoin clients from reaching consensus. To do so, the attacker splits the Bitcoin peer-to-peer network into two disjoint components such that they cannot exchange information. During the attack, nodes in each component will continue to operate normally: they will issue transactions and will add them to a local version of the blockchain. After the attack is mitigated, one of the two versions will be discarded together with the issued transactions and the miners' rewards. As a result, the partition attack allows double-spends and deprives miners of their fair revenue. Importantly, this attack can result in long-term loss of trust in Bitcoin's security followed by a loss to its value from which attackers may benefit [74].

The partition attack is practical and effective today, even against the Bitcoin network as a whole. Its key enabler is the insecurity of today's default Internet routing protocol, BGP, that cannot prevent a malicious AS from diverting traffic through her infrastructure. While intuitive, creating a partition is not trivial. Indeed, some partitions are infeasible as there are connections that cannot be diverted. Even in these cases, the attacker can detect that the desired partition is ineffective and adapt her attack.

The delay attack is a passive attack that compromises the availability of the Bitcoin protocol. To do so, the attacker prevents the victim from securely using Bitcoin by delaying the delivery of new information (blocks) to them. As an intuition, performing such an attack on a merchant leaves the merchant vulnerable to fraud. Indeed, the attacked merchant has no way of knowing whether a client has already spent the coins they use to buy the merchant's goods. Similarly, performing the delay attack on a miner forces them to lose potential revenue of thousands of dollars every ten minutes. Indeed, the attacked miner will work on a block that will likely not be included in the blockchain.

Two key enablers make the delay attack practical and effective today, especially for an attacker targeting particular users. First, the Bitcoin protocol instructs nodes to only request each block from a single one of their peers. As a result, the attacker needs to act only on a single connection. Second, the Bitcoin protocol does not use encryption or any integrity verification to the exchanged traffic. As a result, a network attacker can modify the victim's request, effectively deceiving the requested node into delivering an older block. Notably, the attacker would not be able to undetectably perform the attack by dropping or significantly delaying

a message, as in this case, the transport protocol over which all Bitcoin traffic is routed (i.e., TCP) will terminate the connection.

The perimeter attack is a passive attack that compromises the pseudonymity of the Bitcoin users. Concretely, the attacker maps a victim's transactions to their physical identity by analyzing the propagation pattern of the transactions the victim sends and receives. Such a mapping is dangerous as it often allows the attacker to retrieve the user's entire transaction history [80]. Notably, the existence of multiple transaction surveillance companies that spy on Bitcoin users on behalf of governments, corporations, and individuals, increases the incentives of prospective attackers.

Distinguishing the transaction that a Bitcoin node generated among the thousands it propagates is a known threat that has been exploited by previous deanonymization attacks [73, 40, 39, 42]. To mitigate such attacks, the Bitcoin reference implementation has incorporated diffusion, a privacy-preserving propagation protocol. However, diffusion is oblivious to the underlying network, thus can only protect from application-layer attacks. Indeed, we show that a network-layer attacker can still distinguish the transaction of a targeted node by analyzing a small portion of its traffic using standard anomaly-detection techniques. The two key enablers of this attack are that: *(i)* Bitcoin traffic travels unencrypted; and *(ii)* nodes only receive transactions they ignore; thus, they have not generated themselves.

Besides introducing routing attacks, we also suggest potential defenses for both active and passive attacks. Defending against routing attacks is highly challenging. On the one hand, active attacks exploit a long-standing vulnerability of the Internet functionality. On the other hand, passive attacks are hard to detect, let alone to mitigate.

Mitigating active attacks requires protecting connections from BGP hijacking. While multiple protocols and methods aiming at securing Internet routing exist, they require significant engineering efforts while offering little incentives for early adopters [60, 58, 79]. In effect, the deployment of secure Internet routing protocols has been disappointingly slow.

Considering the harsh operational reality, we present SABRE: an additional secure network (relay) that runs alongside the existing Bitcoin network and can protect the vast majority of Bitcoin clients against a partition attack. SABRE is partially deployable and does not require collaboration or approval from all the Bitcoin or the Internet community. Moreover, SABRE is protected against (D)DoS attacks which is vital for its operations as it is an obvious target for attackers. SABRE achieves these properties through a novel network and node design.

SABRE's network design protects the relay network from BGP hijacking by hosting relays in inherently secure locations that: *(i)* prevent attackers from diverting relay-to-relay connections so as to secure SABRE's internal connectivity; and *(ii)* are attractive (from a routing viewpoint) to many Bitcoin clients so as to secure SABRE's reachability. To achieve this, we leverage a fundamental characteristic of BGP policies: routing attacks cannot divert connections established between two ASes that directly peer with each other and that have no customers. In SABRE, only such ASes are considered for relay locations.

SABRE's node design protects the relay network from (D)DoS attacks by employing programmable networking hardware. Two properties of the Bitcoin protocol enable us to benefit from offloading some of the relay operations to networking hardware. First, most of the relay operations are communication-heavy (propagating information around) as opposed to being computation-heavy. Second, the content (block) that the relays need to propagate each time is predictable and small in size.

Mitigating passive attacks requires bridging the gap between the application and the networking layer. To that end, we suggest a set of comprehensive countermeasures that affect either the Bitcoin protocol itself or its deployment by individual nodes. The former category contains countermeasures that require protocol modifications but are suitable for network-wide deployment. The latter category contains countermeasures that can benefit individual nodes but would not scale to a network-wide deployment e.g., because they rely on third parties.

In summary, the thesis is divided into five parts. In the first part, we provide the needed background information related to both the network and the application layer (Chapter 1). In the second part, we describe the Bitcoin network from the routing perspective and quantify the properties that make routing attacks particularly worrying (Chapter 2). In the third part, we present the three routing attacks in detail and evaluate their effectiveness and practicality through simulation and real-world experiments (Chapters 3, 4, 5). In the fourth part, we describe countermeasures for both active and passive attacks (Chapters 6,7). In the final part, we present our conclusions and sketch future research directions (Chapters 8).

Bibliography

This thesis is based on the following publications:

[1] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," in Proceedings of the 38th IEEE Symposium on Security and Privacy (IEEE S&P'17), 2017.

[2] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: Protecting Bitcoin against Routing Attacks," in Proceedings of the 26th Network and Distributed System Security Symposium (NDSS'19), 2019.

[3] M. Apostolaki, C. Maire, and L. Vanbever, "PERIMETER: A Network-layer Attack on the Anonymity of Cryptocurrencies," Proceedings of the 25th International Conference on Financial Cryptography and Data Security (FC'21), 2021.

[4] Y. Sun, M. Apostolaki, H. Birge-Lee, L. Vanbever, J. Rexford, M. Chiang, and P. Mittal, "Securing Internet Applications from Routing Attacks," Communications of the ACM (CACM), 2021

The remaining publications were part of my PhD research, but they are not covered in this thesis. The topics of these publications are outside the scope of the material covered here.

[5] P. Wintermeyer, M. Apostolaki, A. Dietmüller and L. Vanbever, "P2GO: P4 Profile-Guided Optimizations," in Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets'20), 2020.

[6] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast Connectivity Recovery Entirely in the Data Plane," in Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19), 2019.

[7] M. Apostolaki, L. Vanbever, and M. Ghobadi, "FAB: Toward Flow-aware Buffer Sharing on Programmable Switches," in Proceedings of the 2019 ACM Workshop on Buffer Sizing, 2019.

[8] M. Apostolaki, V. Addanki, M. Ghobadi and L. Vanbever, "FB: A Flexible Buffer Management Scheme for Data Center Switches," in submission.

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Laurent Vanbever for his support and guidance over the years. He was a great role model, taught me to be more systematic and to set ambitious goals. He helped me improve my writing and presentation skills.

I sincerely thank the two other members of my dissertation committee: Prof. Sharon Goldberg and Prof. Srdjan Capkun, for devoting the time to read and provide feedback on my thesis. I want to also deeply thank my collaborators Prof. Aviv Zohar, Prof. Ankit Singla, Prof. Manya Ghobadi, Prof. Jennifer Rexford, Prof. Prateek Mittal, Prof. Alberto Dainotti, Prof. Stefano Vissicchio, Prof. Yixin Sun, Alexander Dietmüller, Thomas Holterbach, Edgar Costa Molero, Vamsi Addanki, Patrick Wintermeyer, Cedric Maire, Gian Marti, Jan Müller, and Henry Birge-Lee. They were all extremely helpful and inspiring. I want to particularly thank Jennifer and Prateek for mentoring me; and Manya for hosting me in Microsoft Research and MIT. Finally, I want to thank Victor Bahl for his support and faith in me.

I thank all the fantastic members of the Networked Systems Group at ETH that made these years so enjoyable and intriguing. I also want to especially thank Coralie, Ege, Tobias, Rudiger for their feedback on my thesis. I am also very grateful to Beat Futterknecht for helping me with all sorts of bureaucratic matters and settling in Switzerland.

I am deeply grateful to all my friends in Zurich for everything we shared during the past five years. I also thank my long-time friends from Crete for the great time during holidays and for helping me through some, especially challenging times. I am grateful to David for the long talks, the short runs, and for his incredible support throughout the years. I also want to thank Sotiris for being my best friend, my greatest supporter, and my beloved partner. Finally, I am deeply thankful to my parents for their unconditional love and support through the years.

Table of Contents

1

Background

In this chapter, we describe the fundamentals of Internet routing (§ 1.1) before introducing the two most widely-used cryptocurrencies, namely Bitcoin and Ethereum (§ 1.2).

1.1 Internet routing

The Internet is a network of over 60k individual networks, known as Autonomous Systems (ASes). ASes build physical connections to each other to exchange traffic under certain business agreements. Oftentimes, ASes also participate in Internet eXchange Points (IXPs) which allow them to exchange traffic with all participating ASes at once. Each AS owns a set of unique IP addresses which it assigns to its customers. IP addresses are grouped in blocks, called IP prefixes.

ASes rely on a distributed protocol, namely BGP [87] to exchange information about how to reach each of the 800k+ IP prefixes [3]. In particular, each AS creates a BGP advertisement of the prefixes it owns. BGP propagates the initial advertisement AS-by-AS to the entire Internet, such that all ASes learn at least one route (AS path) for each destination prefix. BGP is a single-path and policy-based protocol, meaning, each AS *(i)* selects one single best route to reach any IP prefix and *(ii)* selectively exports this route to its neighboring ASes, omitting the AS from which it learned the route.

The selection and exportation processes are governed by the business relationships each AS maintains with its neighbors. The most common business relationships

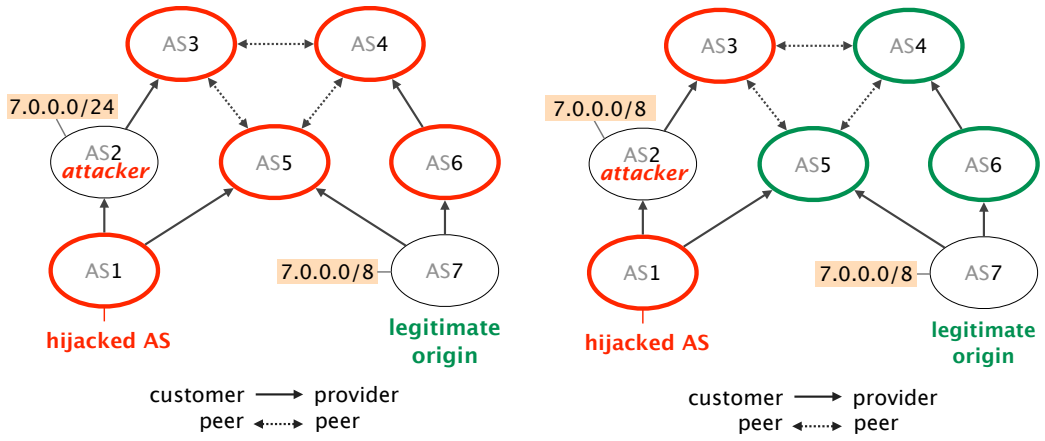
are known as *customer-provider* and *peer-peer* [55]. In a customer-provider relationship, the customer AS pays the provider AS to get full Internet connectivity. In practice, a provider exports to its customer all its best routes and to its neighbors the prefixes advertised by its customers. In a peer-peer relationship, the two ASes connect only to transfer traffic between their respective customers and internal users. Therefore, they only advertise their own prefixes and the routes learned from their customers to each other. Regarding route selection, an AS prefers customer-learned routes over peer-learned ones and peer-learned routes over provider-learned ones. If multiple equally-preferred routes exist (e.g., if two customers announce a route to the same prefix), an AS favors the route with the minimum AS-path length towards the prefix before relying on some arbitrary tie-break [87].

BGP security issues BGP cannot check the validity of route announcements. In effect, a malicious AS can create fake advertisements for any (sub)prefix and advertise them to its neighbors. Such advertisements known as *BGP hijacks* constitute an effective way for an AS to divert traffic directed to given destinations.

We distinguish two types of hijacks according to whether the fake announcement contains: *(i)* a more-specific (longer) prefix than the legitimate one; or *(ii)* an existing (equally specific) prefix. In the former case, the hijacker AS will attract *all* the traffic addressed to the more-specific prefix, independently from its position in the Internet topology. This is because routers forward traffic according to the most-specific prefix matching their destination. In the latter case, the rogue advertisement competes with the legitimate one. The amount of diverted traffic then depends on the relative positions of the attacker and the victim in the Internet topology. Notably, the attacker does not need to pretend to be the origin of the prefix. Instead, she can inject herself in the AS-path.

Fig. 1.1a illustrates an example of a more-specific attack in which AS 2 advertises 7/24, a more-specific prefix of 7/8 which is advertised by AS 7. In doing so, AS 2 effectively redirects the corresponding traffic from *all* ASes except AS 7.[1] In contrast, Fig. 1.1b illustrates the effect of AS 2 advertising 7/8 alongside AS 7. AS 2 only manages to attract the traffic from AS 1 and AS 3. Indeed, AS 1 learns two routes to 7/8 from its two providers (AS 2 and 5) and prefers the illegitimate one from AS 2 because it is shorter. Similarly, AS 3 prefers to reach 7/8 using the customer route learned via AS 2 over the legitimate peer route learned via AS 5.

[1] Traffic from AS7 itself is not redirected as AS7 relies on internal routing protocols, such as OSPF, to reach its own prefixes.



(a) AS 2 hijacks 7/24 **(b)** AS 2 hijacks 7/8

Figure 1.1 The effectiveness of a malicious AS in diverting traffic using BGP hijacks depends on its position and on whether she originates existing prefixes or longer ones. In this example, AS 2 attracts traffic from all ASes when originating 7/24 (a), but only from AS 1 and 3 when originating 7/8 (b).

More-specific hijacks are more powerful but come with drawbacks. First, such attacks are more visible since the hijacked prefixes propagate Internet-wide. In contrast, existing prefixes propagate in smaller regions [61]. For instance, in Fig. 1.1b, while AS 4 and AS 5 learn about the hijacked prefix, they do not propagate it further as they prefer the legitimate announcement. Second, network operators often filter BGP advertisements whose prefix lengths are longer than /24 [72]. This filtering effectively prevents more-specific attacks against existing /24 prefixes.

By default, hijacking a prefix creates a black hole at the attacker's location. However, the attacker can turn a hijack into an *interception* attack and make herself a man-in-the-middle (MITM) by preserving at least one path to the legitimate origin [86, 61]. For instance, in Fig. 1.1a, AS 2 could selectively announce 7/8 to AS 1 so as to keep a working path to the legitimate origin via AS 3. Observe that AS 2 cannot achieve the opposite interception attack, i.e., diverting the traffic from AS 3 and redirecting it to AS 1, as it does not learn a path to the legitimate origin via AS 1. Importantly the attacker does not need to pretend to be the origin of the prefix. Instead, she can inject herself in the AS-path.

1.1.1 Secure routing protocols

To prevent BGP hijacking, researchers and operators have suggested multiple secure routing protocols [43, 59, 66, 84]. We can group these proposals into three categories depending on the guarantees they are offering, namely origin, topology, and path validation. Unfortunately, none of those has seen adequately-wide adoption to secure the Internet [60, 58, 79].

Origin validation The most straight-forward method for securing Internet routing is by preventing malicious ASes from diverting traffic destined to prefixes they do not own. At a high level, origin validation provides a trusted database that binds ASes to the IP prefixes they own. Thus, ASes can verify whether the origin AS of a BGP advertisement adheres to the binding and reject it otherwise.

The most well-known implementation of origin validation is Resource Public Key Infrastructure (RPKI). RPKI is a cryptographic method of signing records that associate a route with an originating AS number. To do so, the RPKI establishes a cryptographic hierarchy of authorities that allocates and suballocates IP address space, as well as authorizes its use in BGP. The RPKI hierarchy is rooted at five Regional Internet registries (RIRs), which allow members to take an IP/ASN pair and sign a Route Origin Authorization (ROA) record. RPKI allows ASes to fetch public repositories that store RPKI objects, cryptographically verify them offline, and apply the resulting mapping of IP prefixes to their authorized origin AS(es) to filter BGP advertisements. Notably, enforcement of origin validation at the core of the Internet, by the top ASes, appears to be both necessary and sufficient for gaining substantial benefits from RPKI[58].

Topology validation While origin validation prevents a malicious AS from advertising a prefix they do not own, nothing prevents them from forging the AS-path in a BGP advertisement while preserving its legitimate origin AS. Topology validation is a method to remedy this threat. To achieve this, protocols such as soBGP [97] maintain a trusted database containing AS-level information. Similar to origin validation, ASes can fetch this topology information to validate whether the advertised AS-path exists. Similarly to RPKI, soBGP is relatively lightweight as it does not require online cryptographic computations or changes to the BGP message structure.

Path validation While deploying origin and topology validation would significantly reduce the risk of BGP hijacks, they do not entirely eliminate the attack vector. In particular, a malicious AS can still attract traffic by announcing a path that exists but is not announced by all ASes in the corresponding path e.g., for financial reasons. Path validation is a method that also protects against

this kind of attack. BGPsec is the most prominent protocol that implements path validation. BGPsec builds on RPKI and adds cryptographic signatures to all BGP messages. BGPsec instructs each AS to digitally sign each of its BGP messages. The signature on a BGPsec message covers the prefix, the AS-level path, the AS number of the AS receiving the BGPsec message, and all the signed messages received from the previous ASes on the path. Unlike origin and topology validation, path validation is heavyweight as it requires not only the deployment of RPKI but also heavy cryptographic operations such as signing and validating BGP messages online. However, BGPsec was only standardized four years ago, and it will likely take many years until it reaches global deployment. Unfortunately, a detailed analysis of the protocol [79] has shown that it performs very poorly unless all ASes use and enforce BGPsec.

1.2 Cryptocurrencies

1.2.1 Bitcoin

Bitcoin is the most widely-used cryptocurrency to date, with over 42 million users in September 2019 [30]. Bitcoin is a decentralized transaction system that relies on a randomized peer-to-peer network to implement a replicated ledger that keeps track of the ownership of funds across Bitcoin addresses. Bitcoin addresses are the equivalent of bank accounts with the difference that they cannot be trivially mapped to the users' real-world identities. Thus, we say that Bitcoin is pseudonymous.

Bitcoin network The Bitcoin system is a peer-to-peer network. Each Bitcoin client uses an IP to connect to multiple other Bitcoin clients. To that end, each Bitcoin client maintains a list of IP addresses of potential peers. The list is bootstrapped via a DNS server, and additional addresses are exchanged between peers through *ADDR* messages. By default, each node randomly initiates 8 *unencrypted* TCP connections to peers in different /16 IP prefixes. Nodes additionally accept connections initiated by others (by default on port 8333). The total number of connections nodes can make is 125 by default. Bitcoin comprises around 10k publicly reachable nodes [22], while ten times more nodes are behind NAT [40]. On top of this peer-to-peer network, the Bitcoin nodes use a consensus mechanism to jointly agree on a (distributed) log of all the transactions that ever happened. This log is called the *blockchain* because it is composed of an ordered list (chain) of grouped transactions (blocks). To that end, the Bitcoin nodes exchange *transactions* and *blocks*.

Transactions transfer value from one address to another. Synchronization is crucial: conflicting transactions attempting to transfer the exact same bitcoins to different destinations may otherwise be approved. Thus transactions must propagate in the entire peer-to-peer network.

To that end, nodes continually listen to transaction announcements which their peers sent via inventory messages (INV). Upon reception of an INV containing a hash of a transaction a node has not heard of, the node replies with a GETDATA message to a *single* peer. This peer then responds by sending the requested information in a TX message. Upon reception of a new transaction, a node verifies it by checking the balance of the corresponding Bitcoin addresses. The node can find this information from data contained in the public Bitcoin blockchain. The node further propagates the verified transaction by advertising to its peers.

The aforementioned propagation mechanism allows an attacker that maintains connections to a large fraction of the Bitcoin clients to trace each transaction back to the node that generated it [73, 42, 39]. Thus, such an attacker can deanonymize all Bitcoin clients. To mitigate this attack vector, the Bitcoin Core has included two modifications on the way transactions are propagated.[2] First, a client advertises transactions with independent, exponential delays to its peers. This broadcast mechanism is called *diffusion* and was introduced as a countermeasure against deanonymization attacks. Second, the diffusion delay that a client adds before an advertisement to a given peer differs depending on which initiated the connection between them. Particularly, a Bitcoin node halves the delay for peers to which it initialized the connection as these are less of a privacy concern [5].

Blocks are batches of transactions that are appended to the blockchain to synchronize the state of the Bitcoin system. Each block contains a cryptographic hash of its predecessor, which identifies its place in the chain, and a proof-of-work. The proof-of-work serves to make block creation difficult and reduces the conflicts in the system. Conflicts, which take the form of blocks that extend the same parent, represent alternative sets of accepted transactions. Nodes converge to a single agreed version by selecting the chain containing the highest amount of computational work as the valid version (usually the longest chain). The proof-of-work also limits attackers' ability to subvert the system. The attackers cannot easily create enough blocks to create a longer alternative chain that would

[2] We mention the modifications that are relevant to our work.

be adopted by the network and result in reversing the transfer of funds (double spend).

The difficulty of block creation is set so that one block is created in the network every 10 minutes on average. Newly created blocks are propagated through the network using a gossip protocol. Most of the time, the 10-minutes interval allows sufficient time for blocks to propagate through the network. Still, two blocks may be created almost concurrently, effectively creating two distinct continuations of the blockchain, namely a fork. The fork is resolved with time as one of the two continuations gradually becomes longer, effectively invalidating the other. The ratio of blocks that are invalidated compared to those created (known as the *orphan rate* or the *fork rate*) expresses the security of the protocol [49, 56, 92] (with lower values being more secure).

Similar to transactions, nodes continually listen to block announcements i.e., INV messages containing the hash of the announced block. If a node determines that it does not hold a newly announced block, it sends a GETDATA message to a *single* peer. The peer then responds by sending the requested information in a BLOCK message. Blocks that are requested and do not arrive within 20 minutes trigger a disconnection from the peer and the block is requested from another peer instead.

Mining pools Mining pools represent groups of miners that divide block creation rewards between them in order to lower the high economic risk associated with mining. They usually comprise of a Stratum server, a gateway and a set of miners [35]. The Stratum server acts as a coordinator and is connected to the gateway and the miners. The gateway is a regular bitcoin node and runs *bitcoind*. The gateway collects recent information regarding newly transmitted transactions and newly built blocks which it then uses to construct a new block template. The gateway sends the template header to the Stratum server, which splits the required work to the miners. The miners attempt to complete the template to a valid block. To do so, miners try different values of the nonce field in the header. If the block is complete, miners send the result back to the Stratum server, which then uses the gateway node to publish the newly formed block to the network. Mining pools often use multiple gateways hosted by different ASes. We refer to the number of different ASes a pool has as its multi-homing degree.

Relay networks are overlay networks maintained by a single administrative entity that run alongside Bitcoin's peer-to-peer network. Relay networks aim at assisting the Bitcoin network, not replacing it. The three most well-known relays are: Falcon [7], FIBRE [8], and the Fast Relay Network (FRN) [16]. These relay networks aim at speeding up block propagation by relying on a system of high-

speed relay nodes which use advanced routing and/or transport-layer techniques. By connecting to these relays, a Bitcoin client can alleviate the effects of bad network performance that may otherwise affect the time needed to acquire a new block.

1.2.2 Ethereum

Ethereum is the second most popular cryptocurrency in market cap [32]. Ethereum has achieved this by supporting decentralized applications that are backed by smart contracts: protocols or small pieces of software running on top of the Ethereum network and performing irreversible transactions with no third-party intervention. Ethereum also offers a digital currency, namely Ether. In the context of Ethereum, a transaction is a data structure describing the exchange of Ether signed with the private key corresponding to a user's pseudonym.

Similar to Bitcoin, Ethereum relies on a peer-to-peer network of nodes and is pseudonymous. In contrast to Bitcoin, though, all Ethereum communications are encrypted [18]. Thus, an on-path eavesdropper cannot read the exchanged messages.

The Ethereum protocol also differs from the Bitcoin protocol in the way it broadcasts transactions. In particular, Ethereum broadcasts newly learned transactions with no delay across transmissions. Ethereum also makes use of an advertisement system, but only for a subset of a node's peers. In particular, consider an Ethereum (Geth [9]) node with n peers, each time it learns a new transaction, the node broadcasts it to $\lfloor \sqrt{n} \rfloor$ of its peers, and then it advertises it to the remaining $n - \lfloor \sqrt{n} \rfloor$ peers, excluding those which are already aware of it.

2

Routing analysis of the Bitcoin network

The first step towards understanding the impact of routing attacks is gaining knowledge about the routing characteristics of the cryptocurrency networks. To that end, we focus on the two most widely-used cryptocurrencies today, namely Bitcoin and Ethereum. We analyze the characteristics of these two overlays when working on top of the Internet underlay and draw unintuitive conclusions. In particular, we find that both the Bitcoin and the Ethereum networks are centralized from the routing perspective, meaning, the distribution of their peer-to-peer network to the Internet is highly skewed. This skewness makes routing attacks extremely practical today as it gives certain ASes significant power over the overlays.

As an intuition, we find that for 50% of the Bitcoin (60% of the Ethereum) clients, there are at least four AS-level adversaries that can intercept 30% of their connections. Moreover, five ASes collectively intercept 72% (80%) of all possible Bitcoin (Ethereum) connections. These characteristics allow an As-level adversary to perform both node-level and network-wide routing attacks as those we describe in the following chapters.

We start by describing the datasets and the techniques we used to infer the routing-augmented Bitcoin and Ethereum topology (§ 2.1). Next, we describe the key findings of our analysis on those typologies and explain how they affect the effectiveness of routing attacks (§ 2.2).

2.1 Methodology

To realistically evaluate the practicality of the routing attacks, we simulated BGP [12], the default routing protocol in the Internet. Particularly, for each pair of ASes in the Internet, we compute the BGP AS path: the sequences of ASes and IXPs that can intercept packets sent by clients hosted in this AS pair. We then calculated the ability of various ASes and IXPs to perform various-powered routing attacks against the Bitcoin and Ethereum networks.

We used four datasets for our evaluation: *(i)* IPs of the Ethereum and Bitcoin clients; *(ii)* advertisement data collected by a well-connected client (supernode); *(iii)* BGP advertised routes; and *(iv)* publicly-available economic relationship among ASes and IXPs. We first describe how we infer the participants of the overlays before we describe how we infer the routing paths among them.

Bitcoin & Ethereum IPs to ASes We fetched the IPs of the Bitcoin and of the Ethereum networks from publicly available data [28, 29]. We removed Bitcoin clients that use Tor (i.e., onion addresses) as we could not assign them to actual IPs. Next, we inferred the most-specific prefix and the AS hosting each Bitcoin and Ethereum client. To that end, we processed almost a million BGP routes (covering all Internet prefixes) advertised on BGP sessions maintained by 6 RIPE BGP collectors [13] (rrc00- rrc05). We then mapped each prefix to the origin AS advertising it.

Bitcoin mining pools Mining pools use regular clients, namely gateways, to connect to the peer-to-peer network. We inferred the IPs of the gateways of each pool in two steps. *First*, we used block propagation information gathered by a supernode connected to ~2000 Bitcoin nodes per day for ten days, assuming that the gateways of a pool are most likely the first to propagate the blocks this pool mines. Particularly, we considered a given IP to belong to a gateway of a pool if: *(i)* it relayed blocks of that pool more than once during the ten-day period; and *(ii)* it frequently was the first to relay a block of that pool (at least half as many times as the most frequent node for that pool). *Second*, we augmented the initial set of IPs with the IPs of the stratum servers used by each mining pool. Indeed, previous studies [81] noted that stratum servers tend to be co-located in the same prefix as the pool's gateway. Since the URLs of the stratum servers are public, we resolved the DNS name (found on the pools websites or by directly connecting to them) and added the corresponding IPs to our IP-to-pool dataset.

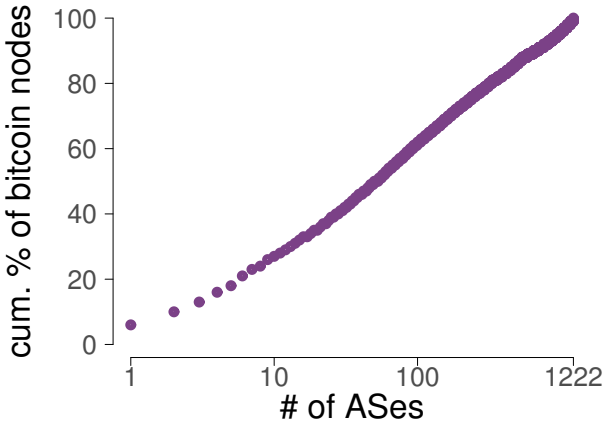


Figure 2.1 Only 13 ASes host 30% of the entire network, while 50 ASes host 50% of the Bitcoin network.

To infer the mining power attached to pools, we tracked how many blocks each pool mined during the ten days interval [4] and assigned them a proportional share of the total mining power.

AS-level topology and forwarding paths To infer an AS-level topology, we used the economic relationships between ASes provided by CAIDA [46]. An AS-level topology is a directed graph in which each node corresponds to an AS, and each link represents an inter-domain connection between two neighboring ASes. Each link is also labeled with the business relationship between the two ASes (customer, peer, or provider). We augmented our AS-level topology with IXP links provided by CAIDA [47] following the methodology in [33, 78].

Our augmented AS-level topology is composed of $\sim 67K$ ASes, more than ~ 700 IXPs, and $\sim 4M$ links. Our datasets were collected in September 2019. We computed the actual forwarding paths on our AS-level topology following the routing tree algorithm described in [61].

2.2 Findings

Using the Internet topology described in § 2.1, we now discuss the key characteristics of the Bitcoin network from the Internet routing perspective. We explain which of them constitute enablers or hindrances for an AS-level attacker. Finally, we explain how these results generalize to the Ethereum network.

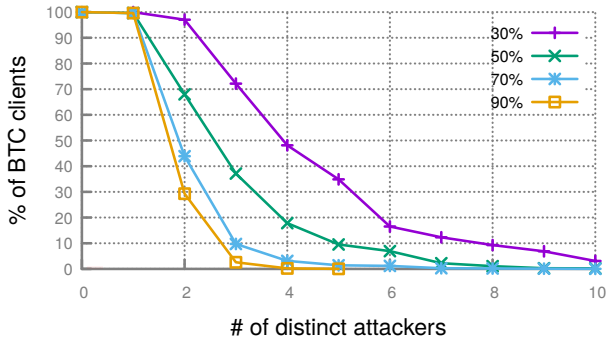


Figure 2.2 For 35% of the Bitcoin clients, there are at least 5 distinct attackers that can intercept 30% of their connections.

A few ASes host most of the Bitcoin nodes and mining power. Fig. 2.1 depicts the cumulative fraction of Bitcoin nodes as a function of the number of hosting ASes. We see that only 13 (resp. 50) ASes host 30% (resp. 50%) of the entire Bitcoin network. These ASes pertain to broadband providers such as Comcast (US), Verizon (US) or Chinanet (CN), as well as to cloud providers such as Hetzner (DE), OVH (FR) and Amazon (US). We observe the same kind of concentration when considering the distribution of Bitcoin nodes per IP prefix. As an intuition, only 63 prefixes (0.0079% of the Internet) host 20% of the network. Worse yet, if we look at mining power, we observe a similar picture. As an intuition, 68% of Bitcoin mining power is hosted in only ten networks.

This high concentration makes Bitcoin traffic easier to intercept and, therefore, more vulnerable regarding the delay and perimeter attacks. With few ASes hosting many nodes, any AS on-path (including the host ASes) is likely to intercept many connections simultaneously, making these attacks more disruptive. The effect of the concentration is a bit more nuanced regarding partition attack. On the one hand, high concentration reduces the total number of feasible partitions because the intra-AS connections cannot be intercepted. On the other hand, the remaining feasible partitions are much easier to achieve since they require fewer hijacked prefixes (§ 3.2).

The majority of connections of most Bitcoin clients traverse a few ASes/IXPs. We calculated the number of distinct attackers able to intercept a fraction of the potential victims' connections. We summarize our results in Fig. 2.2. The x-axis corresponds to the number of distinct attackers that can perform a powerful routing attack against the portion of the Bitcoin clients shown

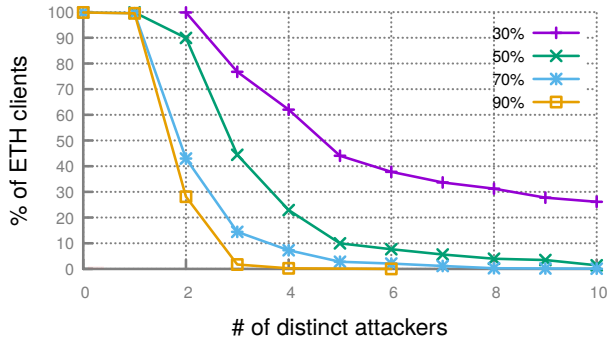


Figure 2.3 For 37% of the Ethereum clients, there are at least 6 distinct attackers that can intercept 30% of their connections.

in the y-axis. We consider four attack types depending on the fraction of the victim's traffic that the attacker intercepts. Specifically, we consider attackers intercepting 30%, 50%, 70%, and 90%, which correspond to different lines in the plots. As expected, all Bitcoin clients are vulnerable to a routing attack by their own provider, which intercepts $> 90\%$ of their connections. Interestingly, $> 90\%$ of all Bitcoin clients are also vulnerable to a routing attack by at least one more network adversary. Moreover, we observe that for 50% of the Bitcoin clients, there are at least 4 attackers able to intercept 30% of their connections.

These findings show the ability of multiple distinct ASes to perform passive attacks against a targeted set of Bitcoin nodes. As an intuition, combining these findings with our analysis on the effectiveness of the delay attack in the actual Bitcoin network (§ 4.3), we can conclude that for half of the Bitcoin clients, there are at least 4 distinct attackers that can delay the propagation of blocks for most of the victims' uptime.

The Ethereum network display similar characteristics, making it also vulnerable to routing attacks. We plot the same results for Ethereum in Fig. 2.3. We observe that Ethereum is slightly more vulnerable than Bitcoin to a passive AS-level (or IXP-level) adversary. Particularly, we observe that for most clients, there are four distinct network adversaries intercepting 30% of their connections. Observe that these adversaries can almost effortlessly infer 30% of the clients' peers.

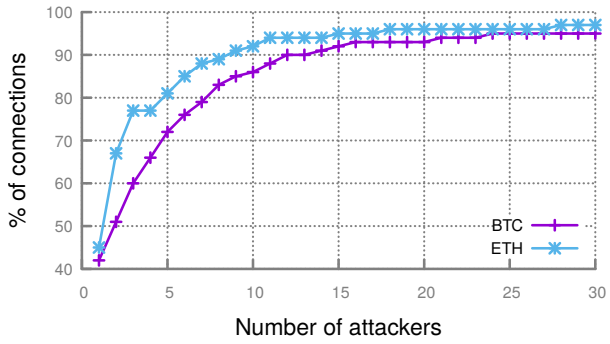


Figure 2.4 5 network adversaries intercept 72% (80%) of all possible Bitcoin (Ethereum) connections.

A few ASes naturally intercept the majority of the Bitcoin traffic, allowing network-wide passive attacks. Fig. 2.4 illustrates the cumulative percentage of connections that can be intercepted by an increasing number of ASes or IXPs (e.g., by colluding with each other). We observe that only ten ASes/IXPs together intercept 90% of the Ethereum clients and 85% of the Bitcoin clients. As an intuition, the list of the most powerful attackers includes well-known ASes such as Amazon, Alibaba, DigitalOcean, and OVHcloud but also well-established IXPs such as the DataIX Novosibirsk, the Amsterdam Internet Exchange, the Hong Kong Internet Exchange, and the London Internet Exchange.

These ASes and IXPs are unlikely to be willing to risk their reputation to attack Bitcoin or Ethereum. Still, for attacks that are passive thus undetectable, this centralization of cryptocurrency is an enabler of network-wide attacks. As an intuition, if those ten network providers decided to collude, they would be able to deanonymize 85% of all transactions in Bitcoin and able to infer at least 90% of the Ethereum peer-to-peer graph. Regarding delay attacks, these few ASes could act as powerful delay attackers. Regarding partition attacks, this observation does not have any direct implication as partitioning requires a *full* cut to be effective (§ 3.2).

>90% of Bitcoin nodes are vulnerable to BGP hijacks 93% of all prefixes hosting Bitcoin nodes are shorter than /24, making them vulnerable to a global IP hijack using more-specific announcements. Indeed, prefixes strictly longer than /24 (i.e., /25 or longer) are filtered by default by many ISPs. Observe that the remaining 7% hosted in /24s are *not* necessarily safe. These can still be hijacked by another AS performing a shortest-path attack, i.e., the attacker,

who will advertise a /24 just like the victim's provider, will attract traffic from all ASes that are closer to her in terms of the number of hops.

While this finding does not have a direct impact on passive attacks, it clearly helps partition attackers as they can divert almost all Bitcoin traffic to their infrastructure using a longer prefix hijack.

Mining pools tend to be distributed and multi-homed, thus more robust. Mining pools have a complex infrastructure compared to regular nodes. We found that *all* pools host their gateways in at least two ASes, while larger pools such as Antpool, F2Pools, GHash.IO, Kano host their gateways in up to five ASes.

Pool multi-homing makes network attacks more challenging, as an AS-level adversary needs to attack every single gateway concurrently. As a result, passive attacks against well-connected mining pools are less effective as the attacker will need to intercept a significant portion of the connections of each gateway. Similarly, active attacks (e.g., partition) are more challenging to perform as the attacker needs to hijack more prefixes. While harder, routing attacks are still possible in the presence of multi-homing.

3

The partition attack

In this chapter, we describe the partition attack. The goal of a partition attack is to completely isolate a set of nodes from the rest of the network. To that end, the attacker first attracts all traffic to the set of nodes she wishes to isolate through her infrastructure. Next, the attacker selectively drops the Bitcoin connections between nodes in the set and the rest of the network.

We first briefly describe the partition attack at a high level and illustrate it using an intuitive example (§ 3.1). We then dive deeper by describing the complete attack procedure with which an attacker can verifiably isolate a selected set of nodes (§ 3.2). Finally, we present a comprehensive evaluation of the attack's practicality and effectiveness (§ 3.3).

We perform a hijack in the wild against our nodes to show that hijacks can effectively divert Bitcoin traffic. We find that it takes less than 90 seconds to divert all traffic flows through the attacker once a hijack is initiated. We also show that any AS in the Internet can isolate up to 47% of the mining power by hijacking less than 100 prefixes, even when considering that mining pools are multi-homed. By analyzing BGP data, we find that hijacks of that many prefixes are frequent in the Internet today and often divert Bitcoin traffic.

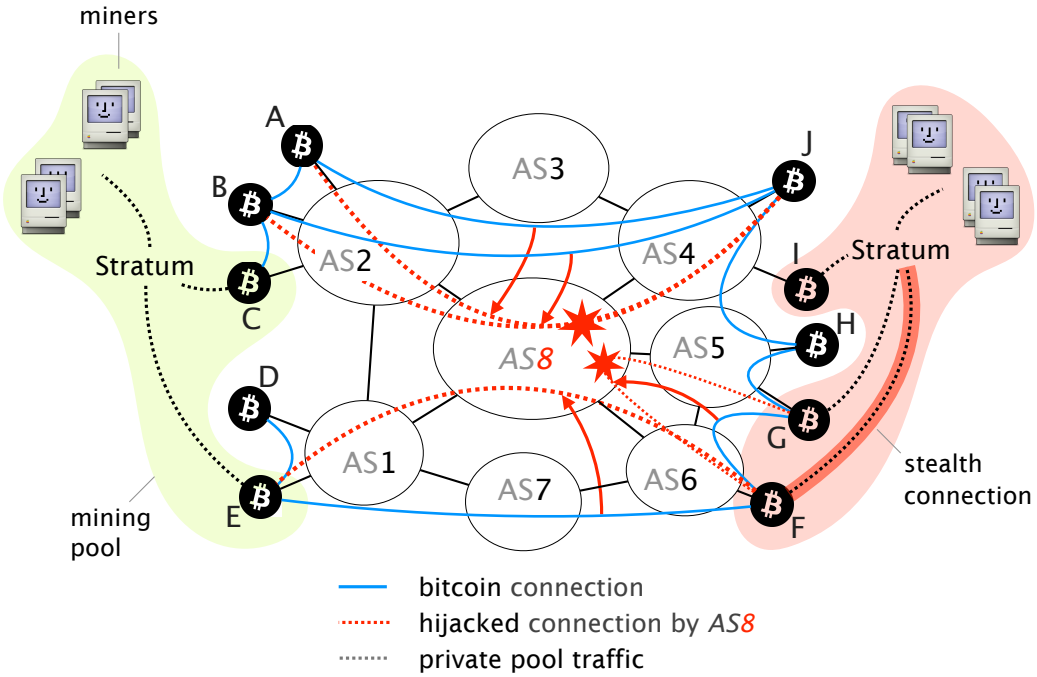


Figure 3.1 Illustration of how an AS-level adversary (AS8) can intercept Bitcoin traffic by hijacking prefixes to isolate the set of nodes $P = (A, B, C, D, E, F)$.

3.1 The partition attack at a high-level

This section gives an overview of the partition attack before we elaborate on its mechanism in Section 3.2. In particular, we first sketch the attacker's goal and procedure. Next, we illustrate the partition attack with an example. Finally, we discuss its impact, practicality, and generality.

In the partition attack, an AS-level adversary seeks to isolate a set of nodes P from the rest of the network, effectively partitioning the Bitcoin network into two disjoint components. The actual content of P depends on the attacker's objectives and can range from single or multiple merchant nodes to a set of nodes that hold a considerable percentage of the total mining power.

The attacker first diverts the traffic destined to nodes in P by hijacking the most-specific prefixes hosting each of the nodes' IP addresses. Once on-path, the attacker distinguishes the Bitcoin traffic (e.g., based on the TCP ports) and identifies whether the corresponding connections cross the partition she tries to create. If so, the attacker drops the packets. If not, meaning the connections are contained within P, she monitors the exchanged Bitcoin messages so as to detect "leakage points". Leakage points are nodes within P maintaining connections

with nodes outside of P, which the attacker cannot intercept, namely "stealth" connections. The attacker can detect these nodes automatically and isolate them from other nodes in P. In Section 3.2, we explain the algorithm the attacker uses to isolate the maximal set of nodes in P that can be isolated.

Example We illustrate the partition attack on the simple network in Fig. 3.1 that is composed of 8 ASes, some of which host Bitcoin nodes. Two mining pools are depicted as a green (left) and a red (right) region. Both pools are multi-homed and have gateways in different ASes. For instance, the red (right) pool has gateways hosted in AS4, AS5, and AS6. We denote the initial Bitcoin connections with blue lines and those that have been diverted via hijacking with red lines. Dashed black lines represent private connections within the pools. Any AS on the path of a connection can intercept it.

Consider an attack by AS8 that is meant to isolate the set of nodes $P = (A, B, C, D, E, F)$. First, it hijacks the prefixes advertised by AS1, AS2 and AS6, as they host nodes within P, effectively attracting the traffic destined to them. Next, AS8 drops all connections crossing the partition: i.e., (A, J), (B, J) and (F, G).

Observe that node F is within the isolated set P, but is also a gateway of the red pool with which F most likely communicates. This connection may not be based on the Bitcoin protocol, and thus, it cannot be intercepted (at least not easily). Thereby, even if the attacker drops all the Bitcoin connections she intercepts, node F may still learn and leak transactions and blocks produced on the other side. Isolating P as such is infeasible. However, AS8 can identify node F as the leakage point during the attack and exclude it from P, essentially isolating $I' = (A, B, C, D, E)$ instead. This I' is actually the maximum subset of P that can be isolated from the Bitcoin network.

Practicality We extensively evaluate the practicality of isolating sets of nodes of various sizes (§ 3.3). We briefly summarize our findings. *First*, we performed a real BGP hijack against our own Bitcoin nodes and show that it takes less than 2 minutes for an attacker to divert Bitcoin traffic. *Second*, we estimated the number of prefixes to hijack to isolate nodes with a given amount of mining power. We found that hijacking only 39 prefixes is enough to isolate a specific set of nodes which accounts for almost 50% of the overall mining power. Through a longitudinal analysis spanning over six months, we found that much larger hijacks happen regularly and that some of them have already impacted Bitcoin traffic. *Third*, we show that, while effective, partitions do not last long after the attack stops: the two components of the partition quickly reconnect, owing to

natural churn. Nevertheless, it takes hours for the two components to be densely connected again.

Impact The impact of a partition attack depends on the number of isolated nodes and how much mining power they have. Isolating a few nodes essentially constitutes a denial of service attack and renders them vulnerable to 0-confirmation double spends. Disconnecting a considerable amount of mining power can lead to the creation of two different versions of the blockchain. All blocks mined on the side with the least mining power will be discarded and all included transactions are likely to be reversed. Such an attack would cause revenue loss for the miners on the side with the least mining power and a prominent risk of double spending. The side with the most mining power would also suffer from an increased risk of selfish mining attacks by adversaries with mining power.

Generalization The partition attack generalizes to other cryptocurrencies and blockchain systems. That is the case as a partition attack prevents nodes from reaching consensus, which is a crucial requirement for the operation of such systems. Of course, the detection and impact of the attack, as well as the exact attack procedure might differ depending on the protocol that each system runs. On the one hand, permissioned blockchain system's (i.e., systems in which the participants are known beforehand) can easily detect that the network is partitioned and seize operations. In this case, a partition attack's impact will be limited to a Denial of Service attack. On the other hand, in permissionless systems, a partition cannot be distinguished from a reduction of the online nodes as nodes independently join and leave the network. In this case, similarly to Bitcoin, the protocol might continue its regular operations resulting in harsher security implications. Similarly, if the traffic of the targeted system is encrypted, the attacker might also need to employ nodes to sense the effectiveness of the attack and adapt if needed.

3.2 A deeper dive into the partition attack

In this section, we elaborate on partition attacks in which an AS-level adversary seeks to isolate a set of nodes P. We first describe which partitions are feasible by defining which connections may cause information leakage (§ 3.2.1) to the isolated set. We then discuss how an attacker may better select a feasible P if she has a partial view of the Bitcoin topology (§ 3.2.2). Next, we walk through the entire attack process, starting with the interception of Bitcoin traffic,

the detection of leakage points, and the adaptation of P until the partition is successfully created (§ 3.2.3). In particular, we present an algorithm that, given a set of nodes P, leads the attacker to isolate the maximal feasible subset. Finally, we prove that our algorithm is correct (§ 3.2.4).

3.2.1 Characterizing feasible partitions

An attacker can isolate a set of nodes P from the network if and only if *all* connections (a, b) where $a \in P$ and $b \notin P$ can be intercepted. We refer to such connections as *vulnerable* and to connections that the attacker cannot intercept as *stealth*.

Vulnerable connections: A connection is *vulnerable* if: *(i)* an attacker can divert it via a BGP hijack; and *(ii)* it uses the Bitcoin protocol. The first requirement enables the attacker to intercept the corresponding packets, while the second enables her to identify and then drop or monitor these packets.

As an illustration, consider Fig. 3.2a, and assume that the attacker, AS8, wants to isolate $P = \{A, B, C\}$. By hijacking the prefixes pertaining to these nodes, the attacker receives all traffic from nodes A and B to node C, as well as the traffic from node D to node A. The path the hijacked traffic follows is depicted with red dashed lines and the original path with blue lines. As all nodes communicate using the Bitcoin protocol, their connections can easily be distinguished, as we explain in Section 3.2.3. Here, AS8 can partition P from the rest of the network by dropping the connection from node D to node A.

Stealth connections: A connection is *stealth* if the attacker cannot intercept it. We distinguish three types of stealth connections: *(i)* intra-AS; *(ii)* intra-pool; and *(iii)* pool-to-pool.

intra-AS: An attacker cannot intercept connections within the same AS using BGP hijack. Indeed, internal traffic does not get routed by BGP but by internal routing protocols (e.g., OSPF, EIGRP). Thus, any intra-AS connection crossing the partition border renders the partition infeasible. Such connections represent only 1.14% of all the possible connection nodes can create (this percentage is calculated based on the topology we inferred in Section 2).

As an illustration, consider Fig. 3.2b and assume that the attacker, AS8, wants to isolate $P = \{A, B, C\}$. By hijacking the corresponding BGP prefixes, AS8 can intercept the connections running between nodes A and B to node C. However, she does not intercept the intra-AS connection between A and X. This means that node X will inform node A of the blocks mined in the rest of the network, and

node A will then relay this information further within P. Thus, $P = \{A, B, C\}$ is not feasible. Yet, observe that isolating $I = \{B, C\}$ is possible. In the following, we explain how the attacker can detect that A maintains a stealth connection leading outside of the partition and dynamically adapt to isolate I instead.

intra-pool: Similar to intra-AS connections, an attacker might not be able to cut connections between gateways belonging to the same mining pool. This is because mining pools might rely on proprietary or even encrypted protocols for internal communication.

As an illustration, consider Fig. 3.2c and assume that the attacker, AS8, wants to isolate $P = \{A, B, C, D\}$. By hijacking the corresponding prefixes, she would intercept and cut all Bitcoin connections between nodes A, B, C, D and nodes E, F. However, nodes A and F would still be connected internally as they belong to the same (green) pool. Again, observe that while isolating $P = \{A, B, C, D\}$ is not feasible, isolating $I = \{B, C, D\}$ from the rest of the network is possible.

pool-to-pool: Finally, an attacker cannot intercept (possibly encrypted) private connections corresponding to peering agreements between mining pools. From the attacker's point of view, these connections can be treated as intra-pool connections, and the corresponding pair of pools can be considered as one larger pool. Note that such connections are different from public initiatives to interconnect all pools, such as the Bitcoin relays [17]. Unlike private peering agreements, relays *cannot* act as bridges to the partition. Indeed, as the IPs of the relays are publicly available, relays are also vulnerable to both hijacking and passive attacks. As such, they can neither bridge the partition nor act as a protected relay when an actual network attack takes place.

3.2.2 Preparing for the attack

In light of these limitations, the attacker can apply two techniques to avoid having stealth connections crossing the partition she creates. First, she can include in P either all or none of the nodes of an AS to avoid intra-AS connections crossing the partition. This can be easily done as the mapping from IPs to ASNs is publicly available [14]. Second, she can include in P either all or none of the gateways of a pool to avoid intra-pool connections crossing the partition. Doing so requires the attacker to know all the gateways of the mining pools she wants to include in P. Inferring the gateways is outside the scope of this thesis, yet the attacker could use techniques described in [81] and leverage her ability to inspect the traffic of almost every node via hijacking. We summarize here two ways hijacking can be used to reveal the gateways of pools.

Algorithm 1 Partitioning algorithm.

Input: - P, a set of Bitcoin IP addresses to disconnect from the rest of the Bitcoin network; and
- $S = [pkt_1, \cdots]$, an infinite packet stream of diverted Bitcoin traffic resulting from the hijack of the prefixes pertaining to P.

Output: False if there is no node $\in P$ that can be verifiably isolated;

```
1  enforce_partition(P, S):
2  begin
3      U ← ∅   L ← ∅   while P \ (L ∪ U) ≠ ∅ do
4          for pkt ∈ S do
5              if pkt.ip_src ∈ P ∧ pkt.ip_src ∉ L then
6                  last_seen[pkt.ip_dst] = now()   U ← U \ {pkt.ip_src}
                   detect_leakage(U, pkt)
7              else
8                  drop(pkt)
9              for src ∈ P ∧ src ∉ L do
10                 if last_seen[src] > now() − threshold then
11                     U ← U ∪ {src}
12     return false
```

1. The attacker may hijack the pool's stratum servers to discover connections that they establish and thus reveal the gateways they connect to. Connections between the pool's stratum server and its gateways are done via bitcoind's RPC access and can be easily distinguished.

2. The attacker may hijack the relay network to discover the connections used by large mining pools. The relay network has indeed a public set of six IPs that pools connect to. By hijacking these, one may learn the IPs of various gateways. Specifically, it will not be hard to identify a pool by observing the blocks each publishes to the relay network.

Even with the above measures, P may still contain leakage points that the attacker will need to identify and exclude (see below). Nevertheless, these considerations increase the chances of establishing the desired partition and reduce the time required to achieve it.

Algorithm 2 Leakage detection algorithm.

Input: - U, a set of Bitcoin IP addresses the attacker cannot monitor; and
- pkt, a (parsed) diverted Bitcoin packet.

```
13 detect_leakage(U, pkt):
14 begin
15     if contains_block(pkt) ∨ contains_inv(pkt) then
16         if hash(pkt) ∈ Blocks(¬(P \ L)) then
17             L ← L ∪ {pkt.ip_src}    drop(pkt)
```

3.2.3 Performing the attack

We now describe how a network adversary can successfully perform a partition attack. The attack is composed of two main phases: *(i)* diverting relevant Bitcoin traffic; and *(ii)* enforcing the partition. In the former phase, the adversary diverts relevant Bitcoin traffic using BGP hijacking. In the latter phase, the attacker cuts all vulnerable connections that cross the partition and excludes from P nodes, which are identified as leakage points. Leakage points are nodes that are connected to the rest of the network via stealth connections.

Intercept Bitcoin traffic: The attacker starts by hijacking all the prefixes pertaining to the Bitcoin nodes she wants to isolate, i.e., all the prefixes covering the IP addresses of nodes in P. As a result, she receives all the traffic destined to these hijacked prefixes, which she splits into two packet streams: relevant and irrelevant. Relevant traffic includes any Bitcoin traffic destined to nodes in P. This traffic should be further investigated. Irrelevant traffic corresponds to the remaining traffic, which should be forwarded back to its legitimate destination.

To distinguish between relevant and irrelevant traffic, the attacker applies a simple filter matching on the IP addresses, the transport protocol and ports used, as well as certain bits of the TCP payload. Specifically, the attacker first classifies as irrelevant all non-TCP traffic and all traffic with destination IPs that are not included in P. In contrast, the attacker classifies as relevant all traffic with destination or source TCP port the default Bitcoin port (8333). Finally, she classifies as relevant all packets which have a Bitcoin header in the TCP payload. Any remaining traffic is considered irrelevant.

Partitioning algorithm: Next, the attacker processes the relevant traffic according to Algorithms 1 and 2. We start by presenting their goal before describing them in more detail.

The high-level goal of the algorithms is to isolate as many nodes in P as possible. To do so, the algorithms identify L, the nodes that are leakage points, and disconnect them from the other nodes in P. Also, the algorithms maintain a set of verifiably isolated nodes $P' = P \setminus \{U \cup L\}$, where U corresponds to the nodes that cannot be monitored (e.g., because they never send packets). In particular, Algorithm 2 is in charge of identifying L, while Algorithm 1 is in charge of identifying U and performing the isolation itself.

We now describe how the algorithms work. Algorithm 1 starts by initializing L and U to $\varnothing$. For every received packet, the algorithm first decides whether the packet belongs to a connection internal to $P \setminus L$ or to one between a node in $P \setminus L$ and an external node based on the source IP address. If the source IP is in $P \setminus L$, the packet belongs to an internal connection, and it is given to Algorithm 2 to investigate if the corresponding node acts as a leakage point (Algorithm 1, Line 6). Otherwise, the packet belongs to a connection that crosses the partition and is dropped (Algorithm 1, Line 8).

Given a packet originated from $P \setminus L$, Algorithm 2 checks whether the sender of the packet is advertising information from outside of $P \setminus L$. Particularly, the attacker checks whether the packet contains an INV message with the hash of a block mined outside of $P \setminus L$ (or the block itself). If it does so, the sender must have a path of stealth connections to a node outside of $P \setminus L$ from which the block was transmitted. Thus the sender is a leakage point and is added to L (Algorithm 2, Line 17). The actual packet is also dropped to prevent this information from spreading.

To detect whether a node in $P \setminus L$ is a leakage point, an attacker should be able to intercept at least one of that node's connections. Specifically, the node should have a vulnerable connection to another node within $P \setminus L$, so that the attacker can monitor the blocks it advertises. To keep track of the nodes that the attacker cannot monitor, Algorithm 1 maintains a set U which contains the nodes she has not received any packets from for a predefined time threshold. (Algorithm 1, Line 11). Whenever one of these nodes manages to establish a connection that the attacker intercepts, it is removed from U (Algorithm 1, Line 6).

Example: We now show how the algorithms work on the example of Fig. 3.2b in which the attacker, AS8, aims to isolate $P = \{A, B, C\}$. By hijacking the prefixes corresponding to these nodes, the attacker intercepts the connections (B, C) and (A, C) and feeds the relevant packets to the algorithms. Recall that the partition is bridged by a stealth (intra-AS) connection between nodes A and X, which cannot be intercepted by the attacker. When a block outside P is mined, node X will inform A, which then will advertise the block to C. The

attacker will catch this advertisement and will conclude that node A is a leakage point. After that, the attacker will drop the packet and will add A to L. As such, all future packets from A to other nodes within $P \setminus L = \{A, B\}$ will be dropped. Observe that the partition isolating $P \setminus L = \{B, C\}$ is indeed the maximum feasible subset of P.

3.2.4 Correctness of the partitioning algorithm

We now prove the properties of Algorithm 1.

Theorem 3.1. *Given P, a set of nodes to disconnect from the Bitcoin network, there exists a unique maximal subset $I \subseteq P$ that can be isolated. Given the assumption that Bitcoin nodes advertise blocks that they receive to all their peers, Algorithm 1 isolates all nodes in I, and maintains a set $P' = P \setminus \{U \cup L\} \subseteq I$ that contains all nodes in I that have a monitored connection and are thus known to be isolated.*

Proof. Consider the set of nodes $\mathcal{S} \subseteq P$ that has a path of stealth connections to some nodes not in P. Clearly, nodes in $\mathcal{S}$ cannot be isolated from the rest of the network by the attacker. Let $I = P \setminus \mathcal{S}$. Notice that I is the maximal set in P that can be disconnected by an attacker. Now, notice that every node in S is placed in sets L or U by the algorithm: if the node has a monitored connection and is caught advertising external blocks it is placed in L (Algorithm 2 Line 17). If it is not monitored then it is placed in U (Algorithm 1, Line 11).

Notice also that the entire set I is isolated from the network. If some node has no stealth connection outside and was removed solely for the lack of monitoring, it is still having all its packets from outside of $P \setminus L$ dropped – Algorithm 1 Line 8).

■

3.3 Evaluation of the partition attack

In this section, we evaluate the practicality and effectiveness of partition attacks by considering four different aspects of the attack. *First*, we show that diverting Bitcoin traffic using BGP hijacks works in practice by performing an actual hijack targeting our own Bitcoin nodes (§ 3.3.1). *Second*, we show that hijacking fewer than 100 prefixes is enough to isolate a large amount of the mining power

due to Bitcoin's centralization (§ 3.3.2). *Third*, we show that much larger hijacks already happen in the Internet today, some already diverting Bitcoin traffic (§ 3.3.3). *Fourth*, we show that Bitcoin quickly recovers from a partition attack once it has stopped (§ 3.3.4).

3.3.1 How long does it take to divert traffic with a hijack?

We hijacked and isolated our own Bitcoin nodes, which were connected to the live network via our own public IP prefixes. In the following, we describe our methodology as well as our findings with regard to the time it takes for a partition to be established.

Methodology We built our own virtual AS with full BGP connectivity using Transit Portal (TP) [90]. TP provides virtual ASes with BGP connectivity to the rest of the Internet by proxying their BGP announcements via multiple worldwide deployments, essentially acting as a multi-homed provider to the virtual AS. In our experiment, we used the `Amsterdam` TP deployment as provider and advertised 184.164.232.0/22 to it. Our virtual AS hosted six bitcoin nodes (v/Satoshi:0.13.0/). Each node had a public IP in 184.164.232.0/22 (.1 to .6 addresses) and could therefore accept connections from any other Bitcoin node in the Internet.

We performed a partition attack against our 6 nodes using BGP hijacking. For this, we used `Cornell`, another TP deployment, as the malicious AS. Specifically, we advertised the prefix 184.164.235.0/24 via the `Cornell` TP. This advertisement is a more-specific prefix with respect to the announcement coming from the `Amsterdam` TP and covers all the IPs of our nodes. Thus, after the new prefix announcement is propagated throughout the Internet, Bitcoin traffic directed to any of our nodes will transit via `Cornell` instead of `Amsterdam`. To mimic an interception attack (Chapter 1), we configured the `Cornell` TP to forward traffic back to our AS. As such, connections to our nodes stayed up during the hijack even though they experienced a higher delay.

We performed the attacks 30 times and measured the time elapsed from the announcement of the most specific prefix until all traffic towards our nodes was sent via the `Cornell` TP.

Diverting Bitcoin traffic via BGP is fast (takes <2 minutes) The results of our experiment are shown in Fig. 3.3a. The main insight is that the attacker received the hijacked traffic very quickly. After only 20 seconds, more than 50% of the connections are diverted. Within 1.5 minutes, all traffic was transiting via

the malicious AS. Thus, attacked nodes are effectively isolated almost as soon as the hijack starts.

We took great care to ensure that our experiments are ethical, meaning that they did not negatively impact the actual Bitcoin network. Indeed, we advertised and hijacked a prefix that was assigned to us by the Transit Portal (TP) project [90]. No other traffic was influenced by our announcements. Also, the isolated nodes were experimental nodes we ran ourselves. Actual Bitcoin nodes that happened to be connected to these were not affected, they just had one of their connections occupied (as if they were connected to a supernode).

3.3.2 How many prefixes must be hijacked to isolate mining power?

Having shown that hijacking prefixes is an efficient way to divert Bitcoin traffic, we now study the practicality of isolating a specific set of nodes P. We focus on isolating sets holding mining power because they are: *(i)* more challenging to perform (as mining pools tend to be multi-homed), and *(ii)* more disruptive as successfully partitioning mining power can lead to the creation of parallel branches in the blockchain.

To that end, we first estimate the number of prefixes the attacker needs to hijack to isolate a specific set of nodes as a function of the mining power they hold. In the following subsection, we evaluate how practical a hijack of that many prefixes is regarding the hijacks that frequently occur in the Internet.

Methodology As described in Section 3.2, not all sets of nodes can be isolated as some connections cannot be intercepted. We therefore, only determine the number of prefixes required to isolate sets P that are feasible in the topology we inferred in Section 2. In particular, we only consider the sets of nodes that contain: *(i)* either all nodes of an AS or none of them; and *(ii)* either the entire mining pool, namely all of its gateways or none of them. With these restrictions, we essentially avoid the possibility of having any leakage point within P, that is caused by an intra-AS or intra-pool stealth connection. However, we cannot account for secret peering agreements that may or may not exist between pools. Such agreements are inherently kept private and their existence is difficult to ascertain.

Hijacking $<$100 prefixes is enough to isolate $\sim$50% of Bitcoin mining power In Table 3.1 we show the number of different feasible sets of nodes we found containing the same amount of mining power (4th and 1st column, respectively). We also include the minimum and the median number of the

Isolated mining power	*min. # pfxes to hijack*	*median # pfxes to hijack*	*# feasible partitions*
6%	2	86	20
7%	7	72	23
8%	32	69	14
30%	83	83	1
39%	32	51	11
40%	37	80	8
41%	44	55	3
45%	34	41	5
46%	78	78	1
47%	39	39	1

Table 3.1 Hijacking <100 prefixes is enough to feasibly partition ~50% of the mining power.

prefixes the attacker would need to hijack to isolate each portion of mining power (2nd and 3rd column, respectively).

As predicted by the centralization of the Bitcoin network (Section 2), the number of prefixes an attacker needs to hijack to create a feasible partition is small: hijacking less than 100 prefixes is enough to isolate up to 47% of the mining power. As we will describe next, hijack events involving similar numbers of prefixes happen regularly in the Internet. Notice that the number of prefixes is not proportional to the isolated mining power. For example, there is a set of nodes representing 47% of mining power that can be isolated by hijacking 39 prefixes, while isolating 30% of the mining power belonging to different pools would require 83 prefixes. Indeed, an attacker can isolate additional mining power with the same number of hijacked prefixes when several pools in the isolated set are hosted in the same ASes.

3.3.3 How many hijacks happen today? Do they impact Bitcoin?

Having an estimate of the number of prefixes that need to be hijacked to partition the entire network, we now look at how common such hijacks are over a 6-months window, from October 2015 to March 2016. We show that BGP hijacks are not only prevalent but also end up diverting Bitcoin traffic.

Methodology We detected BGP hijacks by processing *4 billion* BGP updates advertised during this period on 182 BGP sessions maintained by 3 RIPE BGP collectors [13] (rrc00, rrc01 and rrc03). We consider an update for a prefix p as a hijack if the origin AS differs from the origin AS seen during the previous month. To avoid false positives, we do not consider prefixes that have seen multiple origin ASes during the previous month. We count only a single hijack per prefix and origin pair each day: if AS X hijacks the prefix p twice in one day, we consider both as part of a single hijack.

Large BGP hijacks are frequent in today's Internet, and already end up diverting Bitcoin traffic Fig. 3.4 summarizes our results. We see that there are *hundreds of thousands* of hijack events *each month* (Fig. 3.4a). While most of these hijacks involve a single IP prefix, large hijacks involving between 300 and 30,000 prefixes are also seen every month (right axis). Fig 3.4b depicts the number of Bitcoin nodes for which traffic was diverted in these hijacks. Each month, at least 100 Bitcoin nodes are victim of hijacks[1]. As an illustration, 447 distinct nodes (~7.8% of the Bitcoin network) ended up hijacked at least once in November 2015.

3.3.4 How long do the effects of a partition attack last?

Having investigated the methodology and the relative cost of creating a partition, we now explore how quickly the Bitcoin network recovers from a partition attack. We found out that while the two components quickly reconnect, they stay sparsely connected for a very long time. We first describe the experimental setup. Next, we explain why partitions are not persistent in practice and briefly hint at how an attacker could prolong their lifetime.

Methodology We build a testbed composed of 1050 Bitcoin clients running the default `bitcoind` core (v0.12.1) in testnet mode. Each node runs in a virtual machine connected to a virtual switch and is configured with a different random IP address. Nodes automatically connect to other nodes in the testbed. We enforced a 50%–50% partition, by installing drop rules on the switch, which discard any packet belonging to a connection crossing the partition. Observe that a 50%–50% split is the easiest partition to recover from, as after the attack, the chance that a new connection would bridge the two halves is maximal. We measure the partition recovery time by recording the percentage of connections going from one side to the other in 30-minute intervals.

[1] The actual hijack attempt may have been aimed at other services in the same IP range, still, these nodes were affected and their traffic was re-routed.

Bitcoin TCP connections are kept alive for extended periods. As such, new connections are mostly formed when nodes reconnect or leave the network (churn). To simulate churn realistically, we collected the lists of all reachable Bitcoin nodes [6], every 5 minutes, from February 10 to March 10 2016. For every node i connected in the network on the first day, we measured t_i as the elapsed time until its first disappearance. To determine the probability of a node to reboot, we randomly associated every node in our testbed with a type t_i and assumed this node reboots after a period of time determined by an exponential distribution with parameter $\lambda = \frac{1}{t_i}$. The time for the next reboot is again drawn according to the same distribution. This method produces churn with statistics matching the empirical ones. We repeat each measurement at least five times and report the median value found.

Bitcoin quickly recovers from a partition attack, yet it takes hours for the network to be densely connected again We measured how long it takes for the partition to heal by measuring how many connections cross it, before, during and after its formation (Fig. 3.3b). We consider two different attack scenarios: *(i)* the adversary does not intercept any bitcoin traffic before or after the attack; and *(ii)* the adversary intercepts some connections naturally.

Case 1: The adversary intercepts no traffic after the attack. It takes 2 hours until one fifth of the initial number of connections crossing the partition are established, while after 10 hours, only half of the connections have been re-established. The slow recovery is due to the fact that nodes on both sides do not actively change their connections unless they or their neighbors disconnect.

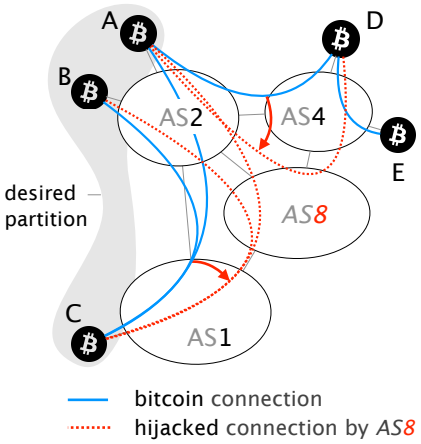
Case 2: The adversary intercepts some traffic after the attack. If an AS-level adversary is naturally on-path for some of the connections, she can significantly prolong the partition's lifetime. To do so, the attacker would just continue to drop packets on connections she naturally intercepts. We measured the effect of such attacks for attackers that are on-path for 14%, 18%, and 28% of the connections, respectively (Fig. 3.3b). We see that an AS-adversary who is initially on-path for 28% of the connections can prolong the already slow recovery of the partition by 58%. Many other ways to increase the persistence of a partition exist.

Intuitively, the attacker needs to suppress the effect of churn in order to keep the victim nodes isolated. The following three observations help in this regard: *Observation 1: A smaller set of nodes will be easier to isolate for extended periods.* The smaller the set of nodes is, the more time will pass before any isolated node restarts. Similarly, if the set of nodes is small, external nodes will connect to the isolated set with lower probability.

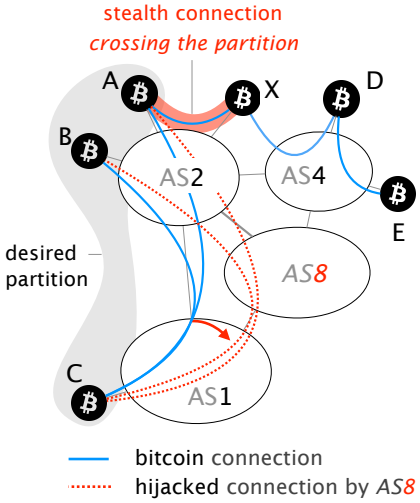
Observation 2: All incoming connection slots can be occupied by connections from attacker nodes. Bitcoin nodes typically limit the number of incoming connections they accept. The attacker may use several nodes to aggressively connect to the isolated nodes and maintain these connections even after BGP routing is restored. Connections initiated by external nodes would then be rejected by the isolated nodes (as all slots of connections remain occupied).

Observation 3: Outgoing connections can be biased via a traditional eclipse attack [65]. Once an isolated node reboots, it will try to establish connections to nodes in its peer lists. These can be biased to contain attacker nodes, other isolated nodes, and IP addresses that do not actually lead to Bitcoin nodes. This is done by aggressively advertising these IPs to the victim nodes. The connections they form upon rebooting will then be biased towards those that maintain the partition.

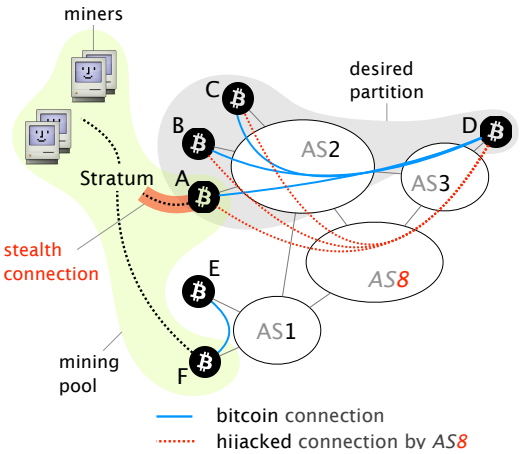
Despite the prolonged healing time, the orphan rate of the network returned to normal even with 1% of all connections crossing the partition. This fact shows that partitions need to be perfect in order to affect the network significantly.



(a) Feasible partition

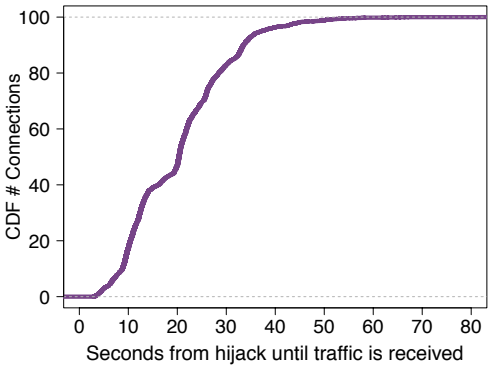


(b) Infeasible partition because of intra-AS connections

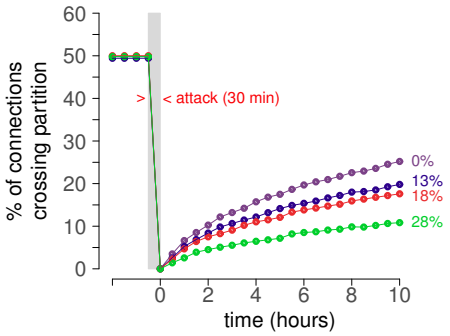


(c) Infeasible partition because of intra-pool connections

Figure 3.2 Not all Bitcoin connections can be diverted by an attacker implying that some partitions cannot be formed.

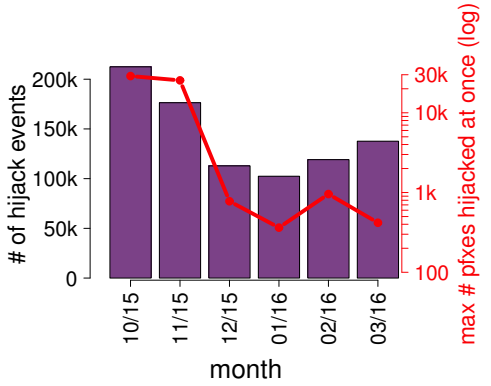


(a) Intercepting Bitcoin traffic using BGP hijack is fast and effective: all the traffic was flowing through the attacker within 90 seconds. Results computed while performing an *actual* BGP hijack against our own Bitcoin nodes.

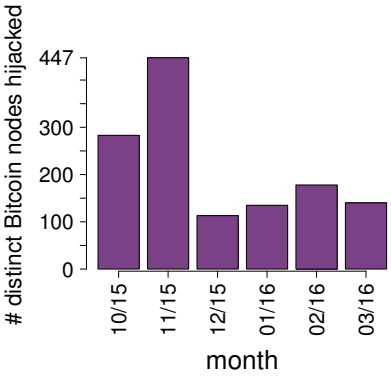


(b) Bitcoin heals slowly after large partition attacks. After 10h, only half as many connections cross the partition. Healing is even slower if the attacker is naturally on-path for 13%, 18%, 28% of the connections.

Figure 3.3



(a) Each month sees *at least* 100,000 hijacks, some of which involve *thousands* of prefixes.



(b) Each month, traffic for at least 100 *distinct* Bitcoin nodes end up diverted by hijacks.

Figure 3.4 Routing manipulation (BGP hijacks) are prevalent today and do impact Bitcoin traffic.

4

The delay attack

In this chapter, we describe the delay attack. The goal of a delay attack is to slow down the propagation of blocks towards or from a given set of nodes. Delaying block propagation negatively affects the security of the protocol [49, 85].

Unlike the partition attack we described in the previous chapter, which requires a perfect cut, the delay attack is effective even when the attacker intercepts a subset of the victims' connections. As such, attackers can perform delay attacks on connections they are naturally intercepting, making the delay attack harder to detect.

We first describe the delay attack at a high level and illustrate it with an example (§ 4.1). We then elaborate on the complete attack procedure, which consists of modifying few key Bitcoin messages while making sure that the connections are not disrupted (§ 4.2). Finally, we present a comprehensive evaluation of the attack's practicality and effectiveness (§ 4.3). In particular, we show that an attacker intercepting 50% of a node's connections can leave it uninformed of the most recent Bitcoin blocks for ~60% of the uptime.

4.1 The delay attack at the high-level

In this section, we give an overview of the delay attack before we elaborate on its workings in Section 4.2. In particular, we first describe the attacker's goal and enablers. Next, we illustrate the delay attack with an example. Finally, we discuss its impact, practicality, and generality.

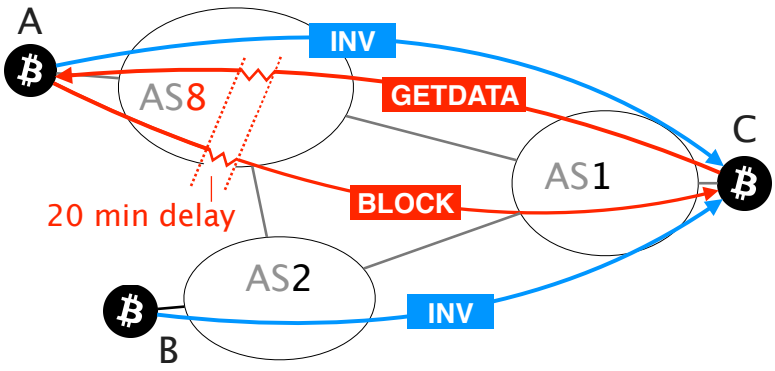


Figure 4.1 Illustration of how an AS-level adversary (AS8) which naturally (i.e., according to BGP) intercepts a part of the traffic can delay the delivery of a block for 20 minutes to a victim node (C).

In a delay attack, the attacker's goal is to slow down the propagation of new blocks sent to a set of Bitcoin nodes without disrupting their connections. As with the partition attack, the delay attack can be targeted, aimed at selected nodes, or network-wide, aimed at disrupting the ability of the entire network to reach consensus [49]. Unlike partition attacks, though, an attacker can delay the propagation of blocks towards a node even if she intercepts a subset of its connections.

Delay attacks leverage three key aspects of the Bitcoin protocol: *(i)* the asymmetry in the way Bitcoin nodes exchange blocks using INV, GETDATA, and BLOCK messages (§ 1); *(ii)* the fact that these messages are not protected against tampering (unencrypted, no integrity checks); and *(iii)* the fact that a Bitcoin node waits for 20 minutes after having requested a block from a peer before requesting it again from another peer. These protocol features enable an attacker, intercepting even one direction of one of the victim's connections, to delay the propagation of a block, as long as this connection is traversed by either the actual BLOCK message or the corresponding GETDATA.

Specifically, if the attacker intercepts the traffic *from* the victim, she can modify the content of the GETDATA message the victim uses to ask for blocks. By preserving the message length and structure and by updating the TCP and Bitcoin checksums, the receiver accepts the modified message, and the connection stays alive. If the attacker intercepts the traffic *towards* a node, she can instead corrupt the content of the BLOCK message such that the victim considers it

invalid. In both cases, the recipient of the blocks remains uninformed for 20 minutes.

Example As an illustration, consider Fig. 4.1 and assume that AS8 is the attacker and C, the victim. Suppose that A and B both advertise a block (say, block X) to C via an INV message and that, without loss of generality, the message from A arrives at C first. C will then send a GETDATA message back to A requesting block X and start a 20-minute timeout count. By modifying the content of the GETDATA node A receives, AS8 indirectly controls what node A will send to node C. This way, the attacker can delay the delivery of the block by up to 20 minutes while avoiding detection and disconnection. Alternatively, AS8 could modify the BLOCK message.

Practicality We verified the practicality of delay attacks by implementing an interception software which we used against our own Bitcoin nodes. We show that intercepting 50% of a node's connections is enough to keep the node uninformed for 63% of its uptime (§ 4.3).

We also evaluated the impact that ASes, which are naturally traversed by a lot of Bitcoin traffic, could have on the network using a scalable event-driven simulator. We found that due to the relatively high degree of multi-homing that pools employ, only very powerful coalitions of network attackers (e.g., all ASes based in the US) could perform a network-wide delay attack. Such an attack is thus unlikely to occur in practice.

Impact Similarly to partition attacks, the impact of a delay attack depends on the number and type (e.g., pool gateway) of impacted nodes. At the node-level, delay attacks can keep the victim eclipsed, essentially performing a denial of service attack or rendering it vulnerable to 0-confirmation double spends. If the node is a gateway of a pool, such attacks can be used to engineer block races and waste the pool's mining power. Network-wide attacks increase the fork rate and render the network vulnerable to other exploits. If a sufficient number of blocks are discarded, miners' revenue is decreased, and the network is more vulnerable to double-spending. A slowdown of block transmission can be used to launch selfish mining attacks by adversaries with mining power.

Generalization While the effects of the delay attack are amplified by the Bitcoin-specific 20-minutes interval and the lack of encryption, the attack still generalizes to other cryptocurrencies. Indeed, the attacker does not need to delay blocks for 20 minutes to impact a cryptocurrency. Notably, increased delay is correlated with deteriorated blockchain security as shown experimentally [49] and theoretically [85]. In fact, the exact impact of any added delay depends,

among others, on the expected interarrival rate of new blocks. As an intuition, in the Ethereum networks, new blocks are mined every 10-20 seconds; as such, even a few seconds of delay will be significant to the system's security. Thus, an AS-level adversary can attack any cryptocurrency whose traffic she intercepts by reordering, dropping, or withholding packets.

4.2 A deeper dive into the delay attack

In this section, we elaborate on the exact procedure of the delay attack that comprises of the adversary modifying the content of specific messages. This is possible due to the lack of encryption and of secure integrity checks of Bitcoin messages. In addition to these, the attacker leverages the fact that nodes send block requests to the first peer that advertised each block and wait 20 minutes for its delivery before requesting it from another peer.

The first known attack leveraging this 20 minutes timeout [57] mandates the adversary to be connected to the victim and to be the first to advertise a new block. After a successful block delay, the connection is lost. In contrast, network-based delay attacks are more effective for at least three reasons: *(i)* an attacker can act on existing connections, namely, she does not need to connect to the victim, which is very often not possible (e.g., nodes behind a NAT); *(ii)* an attacker does not have to be informed about recently mined blocks by the victim's peers to eclipse it; and *(iii)* the connection that was used for the attack is not necessarily lost, prolonging the attack.

Particularly, the effectiveness of the delay attack depends on the direction and fraction of the victim's traffic the attacker intercepts. Intuitively, as Bitcoin clients request blocks from one peer at a time, the probability that the attacker will intercept such a connection increases proportionally with the fraction of the connections she intercepts. In addition, Bitcoin connections are bi-directional TCP connections, meaning the attacker may intercept one direction (e.g., if the victim is multi-homed), both, or none at all. Depending on the direction she intercepts, the attacker fiddles with different messages. In the following, we explain the mechanism that the attacker uses to perform the attack if she intercepts traffic *from* the victim (§ 4.2.1) or *to* the victim node (§ 4.2.2). While in both cases, the attacker does delay block propagation for 20 minutes, the former attack is more effective.

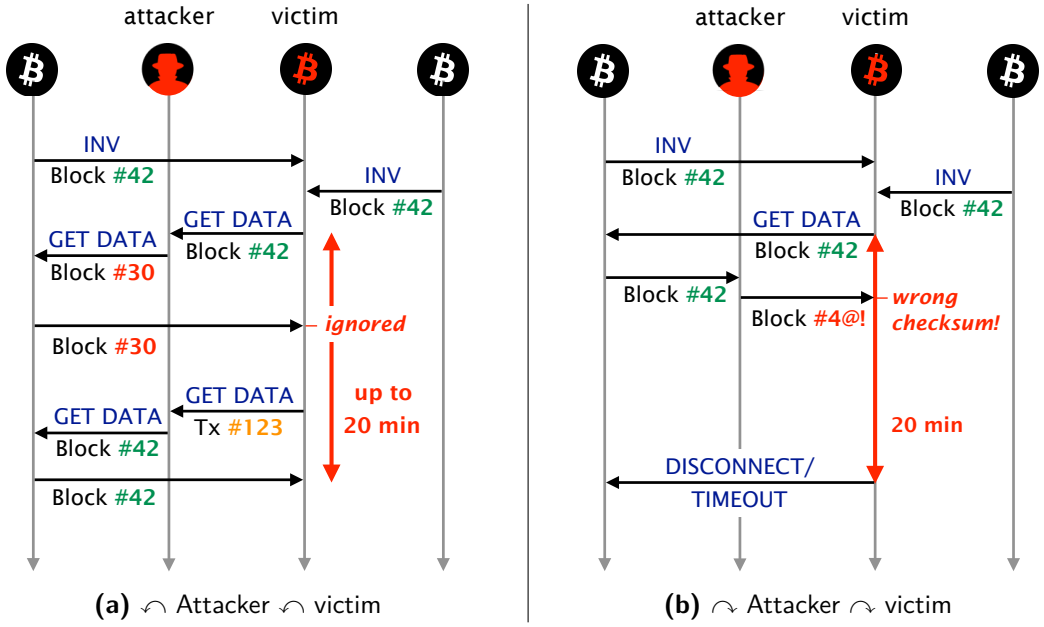


Figure 4.2 An attacker can perform different delay attacks depending on the direction of traffic she intercepts. When on the path from the victim, it can modify the GETDATA message (a), while it can modify the BLOCK message when intercepting the opposite path (b).

4.2.1 The attacker intercepts outgoing traffic

Once a node receives a notification that a new block is available via an INV message, it issues a download request to its sender using a GETDATA message. As an illustration in Fig. 4.2a the victim requests Block 42 from the first of its peers that advertised the block. Since the attacker intercepts the traffic from the victim to this peer, she can modify this GETDATA message to request an older block instead of the original one. In Fig. 4.2a for example, the attacker replaces the hash corresponding to block 42 with that of block 30. The advantage of doing so, over just dropping the packet, is that the length of the message is left unchanged. Notice that if the packet was dropped, the attacker would need to intercept both directions of the connection and update the TCP sequence numbers of all following packets to ensure that the connection is not dropped. If the block is not delivered, the victim node will disconnect after 20 minutes. To avoid disconnection, the attacker can use another GETDATA, sent within the 20-minute window, to perform the reverse operation. Specifically, she modifies the hash back to the original one requested by the victim. Since the GETDATA message for blocks and transactions have the same structure, the attacker is more likely to use the latter as these are much more common. In Fig. 4.2a for

example, she changes the hash of the transaction (Tx #123) to the hash of block 42. Since the block is delivered within the timeout, neither of the nodes disconnects or has any indication of the attack (e.g., an error in the log files).

4.2.2 The attacker intercepts incoming traffic

We now describe the mechanism an attacker would use if she intercepts traffic towards the victim, *i.e.* she can see messages received by the victim, but not the messages that it sends. This attack is less effective compared to the attack working in the opposite direction, as it will eventually result in the connection being dropped 20 minutes after the first delayed block (similarly to [57]). In this case, the attack focuses on the BLOCK messages rather than on the GETDATA. A naive attack would be for the attacker to simply drop any BLOCK message she sees. As Bitcoin relies on TCP, though, doing so would quickly kill the TCP connection. A better yet still simple approach is for the attacker to corrupt the contents of a BLOCK message while preserving the length of the packet (see Fig. 4.2b). This simple operation causes the BLOCK to be discarded when it reaches the victim because of a checksum mismatch. Surprisingly though, we discovered (and verified) that the victim will *not* request the block again, be it from the same or any other peer. After the 20-minute timeout elapses, the victim disconnects because its requested block did not arrive on time.

An alternative for the adversary is to replace the hash of the most recent Block with a hash of an older one in all the INV messages the victim receives. This attack, however, would fail if the attacker intercepts only a fraction of the connections, as the victim will be informed via other connections. This practice is only useful when the attacker hijacks and thus intercepts all the traffic directed to the victim.

4.3 Evaluation of the delay attack

In this section, we evaluate the impact and practicality of delay attacks through a set of experiments at the node level and at the network-wide level. We start by demonstrating that delay attacks against a single node work in practice by implementing a working prototype of an interception software that we then use to delay our own Bitcoin nodes (§ 4.3.1). We then evaluate the impact of network-wide delay attacks by implementing a scalable event-driven Bitcoin simulator. In contrast to partition attacks and to targeted delay attacks, we show that Bitcoin

is well-protected against network-wide delay attacks, even when considering large coalitions of ASes as attackers (§ 4.3.2).

4.3.1 How severely can an attacker delay a single node?

Methodology We implemented a prototype of our interception software on top of Scapy [15], a Python-based packet manipulation library. Our prototype follows the methodology of Section 4.2 and is efficient both in terms of state maintained and processing time. Our implementation indeed maintains only 32B of memory (hash size) for each peer of the victim node. Regarding processing time, our implementation leverages pre-recompiled regular expressions to efficiently identify critical packets (such as those with BLOCK messages), which are then processed in parallel by dedicated threads. Observe that the primitives required for the interception software are also supported by recent programmable data-planes [44] opening up the possibility of performing the attack entirely on network devices.

We used our prototype to attack one of our own Bitcoin nodes (v/Satoshi:0.12.0/, running on Ubuntu 14.04). The prototype ran on a machine acting as a gateway to the victim node. Using this setup, we measured the effectiveness of an attacker in delaying the delivery of blocks by varying the percentage of connections she intercepted. To that end, we measured the fraction of time during which the victim was uninformed of the most recently mined block. We considered our victim node to be uninformed when its view of the main chain is shorter than that of a reference node. The reference node was another Bitcoin client running the same software and the same number of peers as the victim but without any attacker.

Delay attackers intercepting 50% of a node's connection can waste 63% of its mining power Table 4.1 illustrates the percentage of the victim's uptime, during which it was uniformed of the last mined block, considering that the attacker intercepts 100%, 80%, and 50% of its connections. Each value is the average over an attack period of *~200 hours*. To further evaluate the practicality of the attack, the table also depicts the fraction of Bitcoin nodes for which there is an AS, *in addition to their direct provider*, that intercepts 100%, 80%, and 50% of its connections.

Our results reflect the major strength of the attack, which is its effectiveness even when the adversary intercepts only a fraction of the victim's connections. Particularly, we see that an attacker can waste 63% of a node's mining power by intercepting half of its connections. Observe that, even when the attacker is

% intercepted connections	50%	80%	100%
% time victim node is uniformed	63.21%	81.38%	85.45%
% total vulnerable Bitcoin nodes	67.9%	38.9%	21.7%

Table 4.1 67.9% of Bitcoin nodes are vulnerable to an interception of 50% of their connections by an AS *other than their direct provider*. Such interception can cause the node to lag behind a reference node 63.21% of the time.

intercepting all of the victim's connections, the victim eventually gets each block after a delay of 20 minutes.

Regarding the number of vulnerable nodes to this attack in the Bitcoin topology, we found that, for 67.9% of the nodes, there is at least one AS *other than their provider* that intercepts more than 50% of their connections. For 21.7% of the nodes, there is, in fact, an AS (other than their provider) that intercepts all their connections to other nodes. In short, 21.7% of the nodes can be isolated by an AS that is not even their provider.

4.3.2 Can powerful attackers delay the entire Bitcoin network?

Having shown that delay attacks against a single node are practical, we now quantify the network-wide effects of delaying block propagation.

Unlike partition attacks, we show that network-wide delay attacks (that do not utilize active hijacking) are unlikely to happen in practice. Indeed, only extremely powerful attackers such as a coalition grouping all ASes based in the US could significantly delay the Bitcoin network as a whole, increasing the orphan rate to more than 30% from the normal 1%. We also investigate how this effect changes as a function of the degree of multi-homing that pools adopt.

Methodology Unlike partition attacks, the impact of delay attacks on the network is difficult to ascertain. One would need to actually slow down the network to fully evaluate the cascading effect of delaying blocks. We, therefore, built a realistic event-driven simulator following the principles in [83] and used it to evaluate such effects.

Our simulator models the entire Bitcoin network and the impact of a delay attack considering the worst-case scenario for the attacker. Specifically, the delay attacker is only effective if she intercepts the traffic *from* a node that receives a block (essentially if she is able to perform the attack depicted in Fig. 4.2a). An attacker that intercepts the opposite direction is considered unable

to delay blocks during the simulation. Although this choice makes the attacker less effective, it avoids a dependency between the orphan rate and time. Indeed, an attacker intercepting the opposite direction (i.e., to the victim) loses their power through time, as each connection she intercepts is dropped after the first effective block delay and possibly replaced with a connection that the attacker does not intercept. We describe our simulator in detail before we describe our results.

Simulation of the Bitcoin network takes as input a realistic topology and some synthetic topologies with a higher or lower degree of multi-homing. Each of the topologies includes the list of IP addresses running Bitcoin nodes, the IPs of the gateways and the hash rate associated with each mining pool, along with the forwarding paths among all pairs of IPs. We infer the realistic topology as we described in Chapter 2. We generated the synthetic typologies by adding more gateways to the pools in the realistic topology until they all reached the predefined degree of multi-homing.

The simulator models each Bitcoin node as an independent thread that reacts to events according to the Bitcoin protocol. Whenever a node communicates with another, the simulator adds a delay which is proportional to the number of ASes present on the forwarding path between the two nodes. Each node initializes 8 connections following the default client implementation. Blocks are generated at intervals drawn from an exponential distribution, with an expected rate of one block every 10 minutes. The probability that a specific pool succeeds in mining a block directly depends on its mining power. We consider that the gateways of a pool form a clique and communicate with each other with links of zero-delay, which an attacker cannot intercept. Concretely, this means that whenever a pool produces a block, it is simultaneously propagated from all the gateways of this pool. Although this choice makes the attacker less effective, we assume that this is the default behavior of a benign pool.

We verified our simulator by comparing the orphan rate as well as the median propagation delay computed with those of the real network. We found that both of them are within the limits of the actual Bitcoin network [49].

We evaluated the impact of delay attacks considering the three most powerful country-based coalition (the US, China, and Germany) as measured by the percentage of traffic all their ASes[1]

We run the simulation 20 times for each set of parameters and consider a new random Bitcoin topology during each run. In each run, 144 blocks are created,

[1]We map an AS to a country based on the country it is registered in.

Coalition	*Realistic topology* (Section 2)	*Multihoming degree of pools*			
		1	*3*	*5*	*7*
US	23.78	38.46	18.18	6.29	4.20
DE	4.20	18.88	2.10	1.40	1.40
CN	4.90	34.27	1.40	0.70	0.70

Table 4.2 Orphan rate (%) achieved by different network-wide level delay attacks performed by coalitions of *all* the ASes in a country, and considering either the topology inferred in Section 2 or synthetic topologies with various degrees of pool multi-homing. The normal orphan rate is ∼1%.

which is equivalent to a day's worth of block production (assuming an average creation rate of one block per 10 minutes). We evaluated the impact of delay attacks by measuring the orphan rate different adversaries can cause.

Due to pools multi-homing, Bitcoin (as a whole) is not vulnerable to delay attackers, even powerful ones Our results are summarized in Table 4.2. We see that multi-homed pools considerably increase the robustness of the Bitcoin network against delay attacks. In essence, multi-homed pools act as protected relays for the whole network. Indeed, multi-homed pools have better chances of learning about blocks via at least one gateway and can also more efficiently propagate them to the rest of the network via the same gateways.

If we consider the current level of multi-homing, only powerful attackers such as a coalition containing *all* US-based ASes could effectively disrupt the network by increasing the fork rate to 23% (as a comparison, the baseline fork rate is 1%). In contrast, other powerful attackers such as all China-based or all Germany-based ASes can only increase the fork rate to 5%. As such coalitions are unlikely to form in practice, we conclude that *network-wide* delay attacks do not constitute a threat for Bitcoin.

Even a small degree of multi-homing is enough to protect Bitcoin from powerful attackers. If all mining pools were single-homed, large coalitions could substantially harm the currency. The US, for instance could increase the fork rate to 38% while China and Germany could increase it to 34% and 18% respectively. Yet, the fork rate drops dramatically as the average multi-homing degree increases. This is good news for mining pools as it shows that even a small increase in their connectivity helps tremendously in protecting them against delay attacks.

5

The perimeter attack

In this chapter, we introduce the perimeter attack. The goal of the perimeter attack is to deanonymize a particular victim node, meaning to map the victim's IP to their transaction(s) in the Bitcoin network. Such an attack is very harmful to the victim because it often allows the attacker to retrieve the victim's entire transaction history [80].

The perimeter attack is harder to be detected than the partition and even the delay attacks. Unlike the partition attack we described in Chapter 3, the perimeter attacker does not need to forge BGP to perform the attack as she does not need to intercept all connections of the victim to be effective. Unlike the delay attack described in Chapter 4, the attacker does not need to perform online operations (e.g., modify packets at line rate).

We first describe the attack at a high level using a comprehensive example (§5.1) before we elaborate on the attack procedure (§5.2) and describe our evaluation (§5.3). We experimentally show that an adversary intercepting only 25% of the victim's connections can deanonymize it with 70% accuracy. Our routing analysis of the Bitcoin network reveals that such attacks are practical in today's Internet. Indeed, we found that at least 6 distinct network adversaries can independently deanonymize more than 35% of the Bitcoin clients.

5.1 perimeter at a high-level

This section gives an overview of the perimeter attack before we elaborate on its workings in Section 5.2. In particular, we first describe the attacker's goal, profile, and procedure. Next, we illustrate the perimeter attack against a Bitcoin client with an example. Finally, we describe how the attack generalizes to Ethereum.

The attacker's goal is to compute a set of transactions that contains (most of) the victim's transaction(s) (i.e., maximize true positives) and as few as possible transactions created by other nodes (i.e., minimize false positives). We refer to this set of transactions as the victim's *anonymity set*.

As in the delay attack (§ 4), the attacker is an Autonomous System (AS) or Internet eXchange Point (IXP) that naturally (i.e., according to BGP's calculations) intercepts X% of the victim's connections and knows the victim's IP. Two key reasons makes this attacker model practical. First, due to the centralization of the Bitcoin traffic, multiple ASes and IXPs intercept a large portion of a host's connections even if they are not their direct provider, as we show in Section 2.2. Second, finding the IP of a person is practical as it is revealed every time this person visits a website or an application e.g., Skype call.

The attack consists of eavesdropping on the victim's connections and analyzing collected data to distinguish the victim's transaction(s). Concretely, the adversary first leverages her position in the Internet to eavesdrop on the transactions that the victim propagates. We refer to this process as *surrounding* since the adversary creates a logical circle around the victim across which she can observe the incoming and outgoing information. Notably, the adversary surrounds the victim in a purely passive and undetectable manner, as she only monitors traffic that she already forwards. Next, the adversary computes statistics on the transactions the victim advertises and uses anomaly detection to find the victim's transactions. Useful statistics include the number of times the victim or its peers sent or received a transaction.

Example Fig. 5.1a illustrates how an attacker running perimeter can deanonymize Bitcoin transactions. This network comprises seven ASes (AS0 - AS6), some of which host Bitcoin clients (nodes A-G). Traffic between each pair of nodes is forwarded following the AS path that BGP calculates. As a result, AS2 intercepts the connections between node A and nodes B, C, D, and E. Assume that AS2 is malicious and aims at deanonymizing Alice's transactions. AS2 knows the IP of the node on which Alice runs her Bitcoin wallet, namely the IP of node A. Thus, AS2 aims at mapping node A to the transaction(s) it generates, TX#33 in this example.

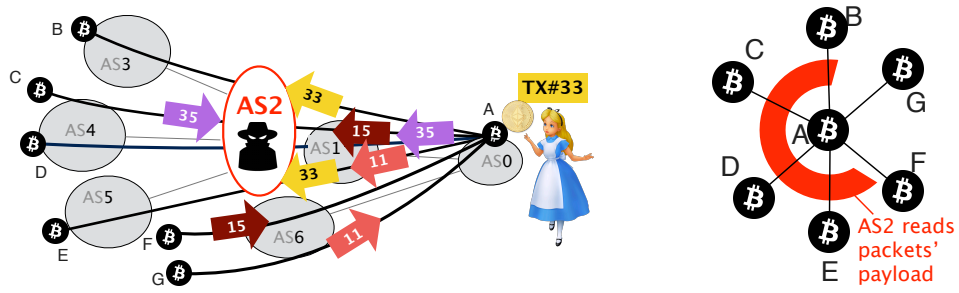


Figure 5.1 (a) From the networking viewpoint, the attacker (AS2) naturally i.e., according to BGP, intercepts some of the victim's connections. (b) From the application viewpoint, the attacker (partially) surrounds the victim without establishing any new connection. Surrounding the victim allows the attacker (AS2) to read the unencrypted Bitcoin messages the victim node A sends and receives.

AS2 eavesdrops on the victim's connections that she naturally intercepts and creates the initial anonymity set from the transactions that node A propagates i.e., $\#15$, $\#11$, $\#35$, and $\#33$. From the application viewpoint, AS2 has passively formed a (partial) logical circle around the victim, as illustrated in Fig. 5.1b. Next, AS2 tries to reduce the size of the anonymity set by removing transactions that are most likely not generated by the victim. AS2 knows that a Bitcoin node only receives transactions it requests and only requests transactions it does not know already. Thus, AS2 excludes $\#35$ from the anonymity set as AS2 has observed node A receiving $\#35$ from node C. AS2 cannot use the same technique for $\#15$ and $\#11$ because AS2 does not intercept the victim's connections to the nodes from which it received these transactions, i.e., nodes F and G. Instead, AS2 uses anomaly detection to find the victim's transaction, which appears as an anomaly with respect to its propagation pattern. For instance, the number of peers that requested $\#33$ from node A is higher for $\#33$ than for any other transaction. We elaborate on the anomaly detection procedure and the features used in §5.2.

Practicality Unlike the partition attack, which requires the attacker to control *all* connections of a victim, perimeter works with just a fraction. We verified the practicality of the perimeter attack by training an isolation forest in simulation and using it against our own Bitcoin node connected to the live Bitcoin network. Using this setup, we found that an adversary intercepting only 50% of the victim's connections can deanonymize it with 90% accuracy (§ 5.3). Notably, we observe

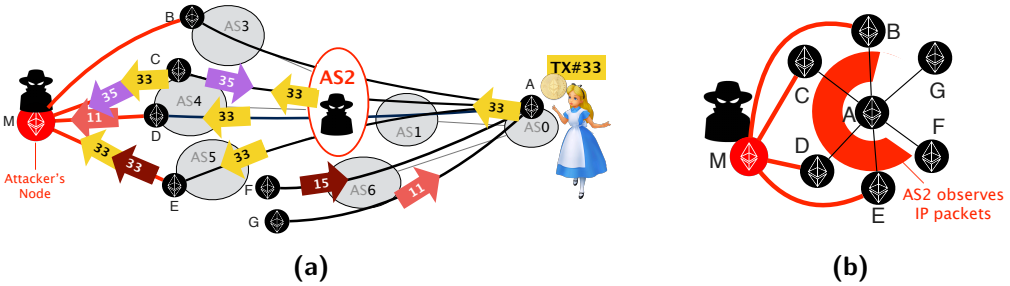


Figure 5.2 To deanonymize Alice's transaction in Ethereum, the attacker (AS2) connects to some of the victim's peers. AS2 infers some of the victim's peers' IPs by eavesdropping on the victim's connections. (b) In effect, the attacker indirectly surrounds the victim from the application viewpoint.

that for most Bitcoin clients, there are four distinct adversaries intercepting 30% of their connections, as we show in Section 2.2.

Impact The impact of a perimeter attack is worrying for users. By mapping a user's transaction to their real-world identity, an attacker can often retrieve the user's entire transaction history [80]. Notably, this is true even if the user follows best practices, i.e., changes her pseudonym frequently. The attack is particularly worrying as the attacker has clear monetary incentives to perform it. Remarkably, multiple surveillance companies [19] spy on Bitcoin users on behalf of governments and other individuals and would thus be likely to buy such data.

Generalization Fig. 5.2a illustrates the same attack scenario as before but with nodes A-G belonging to the Ethereum network. We now explain how AS2 could use perimeter to deanonymize node A in this case. Unlike Bitcoin, Ethereum connections are encrypted, meaning that AS2 cannot directly read the content of the messages the victim exchanges with its peers. To deanonymize an Ethereum client, AS2 uses the observation made by Biryukov et al. [39], according to which a node can be uniquely identified in a single session by its directly connected neighboring nodes. Contrary to the attack presented by Biryukov et al. [39] that used a Bitcoin-specific flaw to infer connections, the perimeter attack can infer the IP addresses of the victim's peers by reading the unencrypted headers of the packets the victim node A inevitably sends and receives. After connecting to some of the victim's peers[1], distinguishing the victim's transactions (across those its peers propagate) is strictly more straightforward than for Bitcoin. That is the case as most Ethereum nodes (geth version [9]) advertise the new transactions

[1]Ethereum facilitates connecting to a client using its IP (i.e., discovery v4 UDP packet).

immediately to their peers. The adversary has again partially surrounded the victim from the application viewpoint, as seen in Fig. 5.2b.

5.2 A deeper dive into the perimeter

Having described the perimeter attack at a higher level in §5.1, we now elaborate on perimeter 's technical details. Concretely, we describe how the attacker *(i)* distinguishes Bitcoin traffic (§5.2.1); *(ii)* retrieves propagated transactions (§5.2.2); and *(iii)* uses anomaly detection (an Isolation Forest) to find the victim's transaction(s) (§5.2.3). Finally, we discuss the features the attacker uses (§5.2.4).

5.2.1 Recognizing Bitcoin traffic

The adversary surrounds the victim node and reads the data exchanged in the Bitcoin connections to create the initial anonymity set. To do so, the adversary first needs to distinguish Bitcoin traffic across all the connections she intercepts. The adversary can easily distinguish Bitcoin traffic since most clients use a particular TCP port, i.e., 8333. For clients using a different TCP port, the adversary can still recognize the Bitcoin connections. To do so, the adversary can search on the packet payload to find known Bitcoin message types, e.g., “inv" or “getdata". Indeed, the adversary can perform string searching at line-rate even in commodity hardware [67]. Notably, the adversary performs a string search on a single packet per connection. Once she finds a Bitcoin message in a packet of a connection, she can use a filter that matches the 4-tuple of the TCP connection (i.e., IP addresses and TCP ports) to distinguish it.

5.2.2 Creating the initial anonymity set

To create the initial anonymity set, the attacker needs to distinguish all transactions that the victim itself or its peers have advertised. This is challenging as Bitcoin messages can be split among multiple packets, and those packets can be re-ordered, lost, and re-transmitted while being transferred in the Internet. As a result, concatenating each Bitcoin connection's payloads (packet stream) would not result in the complete list of messages that the corresponding clients exchanged (message stream). To reconstruct the message stream, the adversary

can use tools such as GoPacket [11] that leverage the sequence number contained in the TCP header.

Next, the adversary includes in the anonymity set the hashes of the transactions that are included in three types of messages, namely “inv", “getdata", and “tx". Finally, the adversary calculates statistics per transaction hash. Particularly, she calculates the number of “inv", “getdata", and “tx" that are sent and received per transaction.

5.2.3 Analyzing data

Having collected the initial anonymity set, the adversary needs to reduce it to the transactions that the victim created. Doing so is challenging for two reasons. First, the number of transactions the victim propagates is orders of magnitude higher than those that it creates. Second, the adversary does not have ground truth to train on (e.g., transactions that the victim created).

To address these challenges, the adversary formulates the problem to an unsupervised anomaly detection problem, meaning a problem that requires identifying data points that differ from the norm (i.e., anomalies) in an unlabeled dataset. Doing so allows the attacker to train directly on the traffic she observes, leveraging the fact that the victim's transactions are a tiny minority compared to all transactions the victim propagates. As a result, the attacker can learn the most common propagation pattern and distinguish the victim's transactions as anomalies. Indeed, the victim's transactions will exhibit different propagation patterns, e.g., the victim will propagate the transaction it generates to more peers compared to other transactions.

The attacker uses an Isolation Forest (IF) ([76],[77]) to solve this unsupervised anomaly detection problem since IF is more efficient, expressive, and interpretable than clustering-based approaches or neural networks. Concretely, IF is more efficient, especially with high-dimensional data, than distance-based methods, including classical nearest-neighbor and clustering-based approaches. This is because IF is a tree-based machine learning algorithm that directly identifies anomalies by isolating outliers in the data rather than first defining the normal behavior and calculating point-based distances. Finally, in contrast to neural network methods, such as autoencoders, IF is easy to interpret and is not too sensitive to parameter tuning. IF achieves this by building an ensemble of decision trees to partition the data points. To create these trees, IF recursively generates partitions by randomly selecting a feature and then selecting a random split value between the minimum and the maximum value of the selected feature.

The number of required random splits to isolate a sample averaged over a forest of such random trees determines the normality of a sample. IF leverages the observation that anomalies are more natural to isolate, and thus they need fewer splits on average than normal data points.

5.2.4 Feature selection

We started our feature investigation with a pool of features, some of which were related to timing and some to interactions between the victim and its peers. Using cross-validation in our simulation runs (§ 5.3.1), we selected three interaction-related features: *(i)* number of "getdata" messages; *(ii)* number of "tx" messages; and *(iii)* the portion of clients which requested a transaction. Next, we describe the selected features in detail and explain why they allow the victim's transaction(s) to stand out as anomalies.

The number of "getdata" messages that the victim received per transaction: This is equivalent to the number of times the victim sent a transaction. Thus, the adversary can capture this feature independently of the direction of the victim's connections she intercepts. A Bitcoin client will only send a "getdata" for an advertised transaction if it has not received this transaction before. Thus, the victim is expected to receive more "getdata" for a transaction it created, as its peers are unlikely to have received it from others.

The number of "tx" messages the victim received per transaction: If the victim received a transaction from one of its peers, then the victim could not have created it. That is because, in order for the victim to receive a transaction, it should have requested this transaction from its peer, and thus, it should not have known this transaction beforehand. The number of "tx" the victim received for a transaction is equivalent to the number of "getdata" the victim sent. In effect, an AS-level (or IXP-level) adversary would be able to calculate this feature independently of the direction of traffic she intercepts, namely *to* or *from* the victim node.

The portion of clients that requested a transaction from the victim across those the victim advertised this transaction to: This feature is similar to the number of "getdata" with one critical difference. It considers that because of diffusion, the victim might delay advertising its transaction to some of the peers so much that they learn it from elsewhere. The victim's transaction will have a high request/advertisement ratio because the victim knows about the transaction much earlier than its peers.

5.3 Evaluation of the perimeter attack

We evaluate the effectiveness of perimeter in simulation (§5.3.1) and in the wild (§5.3.2). We found that an attacker intercepting 25% of the victim's connections can deanonymize a client with 70% accuracy in the Bitcoin Mainnet. Unsurprisingly, the perimeter attack appears even more effective in simulation.

5.3.1 How accurate is the perimeter attack in simulation, taking Internet delays into consideration?

We evaluated perimeter using a realistic simulation whose delays we have tuned based on Internet-wide measurements. We elaborate on our methodology before we describe our results.

Methodology We modeled the entire Bitcoin network by extending the realistic event-driven simulator we described in Section 4.3. We used the 0.19.1 version of the Bitcoin Core as a reference for the behavior of the Bitcoin clients. Among other implementation details, we simulated diffusion: the Poisson delay that a node waits before advertising a new transaction to each peer and the preference to outgoing connections in advertising and requesting transactions [5]. We simulated all nodes whose IPs were reachable, and we could locate in the Internet as described in Chapter 2.

To realistically model Internet delays among clients in our simulation, we leveraged the RIPE Atlas platform. RIPE Atlas [1] is a data collection system composed of a global network of probes, that can actively perform Internet measurements. In particular, to estimate the delay between each pair of Bitcoin nodes, we measure the delay between probes in the ASes hosting these Bitcoin nodes. Indeed, Internet delay between two particular hosts located in any AS-pair is representative of the delay between any pair of hosts in the same AS-pair. That is because the Internet path between any pair of hosts in the same AS-pair is common. We performed ping measurements for each pair of ASes say (ASA, ASB) in which there are at least two Bitcoin clients (e.g., one Bitcoin client in ASA and one in ASB) and at least one RIPE probe available (i.e., either in ASA or in ASB). If multiple probes existed in the same AS, we used one for each prefix in which at least one client is hosted. We perform each measurement at least three times and use the median delay. Our measurement campaign lasted 7 hours and included $\sim 50K$ pings. We leveraged delay measurements available from RIPE atlas [2] to add the delays of AS-pairs, which we could not measure ourselves. Together these delay measurements cover 72% of the Bitcoin connections.

Simulation	25%	50%	75%	100%
true positives	1	1	1	1
false positives	0.002%	0.002%	0.001%	0%

Table 5.1 In simulation, an adversary deanonymizes the victim with 100% accuracy even when intercepting 25% of its connections.

Mainnet	25%	50%	75%	100%
true positives	0.7	0.9	0.9	1
false positives	0.002%	0.003%	0.003%	0.0%

Table 5.2 In the wild, an adversary deanonymizes the victim with 90% accuracy when intercepting 50% of its connections.

We configure the delay of each node pair in the simulation with a randomly-selected value across the delays measured in the corresponding AS-pairs. We validated our augmented simulator by ensuring that the median transaction propagation delay aligns with the value reported in [31].

We simulated a total of 10000 transactions, among which the victim client created 100. We use 70% of all transactions for training and 30% for testing. For feature selection, we use 5-fold cross-validation on the training set. We run the attack assuming the adversary intercepts a fraction of the victim's connection, i.e., 25%, 50%, 75%, 100%.

An attacker intercepting just 25% of the victim's connections can deanonymize it. We summarize our results in Table 5.1. We observe that an attacker can always (i.e., almost independently of the percentage of connections she intercepts) deanonymize the victim with 100% true positive and low false-positive rate. This is expected because the simulated environment is idealized. In particular, *all* clients run the same code, are benign, and are in the same condition concerning load. Such an environment creates a straightforward case for anomaly detection.

5.3.2 Is the perimeter attack accurate in the wild?

We evaluated perimeter on the actual Bitcoin network. We describe our methodology before we describe our results.

Methodology For our in-the-wild experiment, we used as victim a Bitcoin node version 0.19.1 of the Bitcoin Core running. Since we only attack our own node, our experiment is ethical: we did not disturb the normal operation of the Bitcoin network in any way. We configured our victim to not listen for incoming connections but instead only connect to a predefined set of peers randomly selected across those in [29].[2] We capture a total of $\sim 30K$ of transactions, among which 10 transactions were created by our victim. As the attack is completely passive, we use the same transactions to measure the effectiveness of various powered adversaries. In particular, we run the attack assuming the adversary intercepts a fraction of the victim's connection i.e., 25%, 50%, 75%, 100%.

We split the resulting dataset into the training and testing sets. We used all the victim's transactions and 30% of the transactions from other clients as the testing set. We included all of the victim's transactions in the testing to have a more accurate estimate of true positives. In any case, the victim's transactions are too few to affect the training of the model. We used the remaining 70% of transactions from other clients as the training set. Finally, we used the features we selected in the simulation, and we describe them in §5.2.4.

The perimeter attack is more accurate than previously known deanonymization attacks. Table 5.2 summarizes our results. We observe that an adversary intercepting only 25% of the victim's connections can deanonymize it with 70% true positives and only 0.002% false positives. Moreover, an adversary intercepting 50% of a client's connections (or more) can deanonymize the victim with 90% accuracy (or above). As a baseline, consider that previous attacks using "supernodes" report accuracies of 11%-60% [39] and 75% [42], thus lower than perimeter . This demonstrates the effectiveness of a network-layer attacker exploiting her access over the Internet infrastructure.

[2]We do not allow incoming connections to prevent attacks from light clients during the experiment.

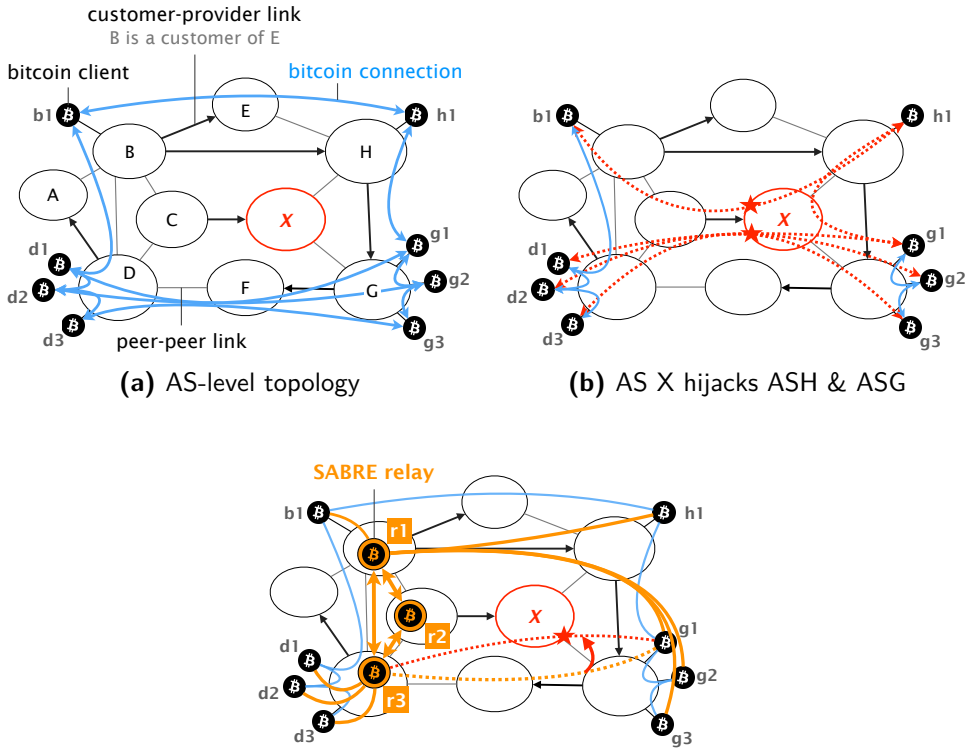
6

Defenses against active attacks

In this chapter, we introduce SABRE, a relay network that can protect the Bitcoin network from active attacks, such as the partition attack, which we describe in Chapter 3. The SABRE relay is a network of nodes, independent from the actual Bitcoin network that provides Bitcoin clients with an additional secure channel for exchanging information, even in the presence of an AS-level attacker that hijacks Bitcoin traffic.

We start with a high-level description of how SABRE protects the Bitcoin network from a particular attacker model (§ 6.1) that combines partition with a DDoS attack. Next, we elaborate on the two key design strategies on which SABRE relies. First, we explain SABRE's network design, meaning how SABRE selects the Internet location of its relay nodes to prevent attackers from disrupting relay-to-relay, node-to-relay and relay-to-node connections (§ 6.2). Second, we explain SABRE's node design, meaning how SABRE nodes are implemented to sustain the expected load (§ 6.3). Notably, we also discuss SABRE 's viable deployment scenarios and show that it is partially deployable (§ 6.4).

Finally, we evaluate SABRE's effectiveness and practicality. Using actual routing data, we show that SABRE deployments with six nodes are already enough to protect 80% of the clients from 96% of the AS-level adversaries (§ 6.5). We show that SABRE is practical in today's hardware technology by implementing it on a programmable switch and connecting it to an extended regular Bitcoin client (§ 6.6).



(a) AS-level topology

(b) AS X hijacks ASH & ASG

(c) With SABRE, network stays connected

Figure 6.1 SABRE protects the Bitcoin network from AS-level adversaries aiming to partition it. Without SABRE, AS X can split the network in half by first diverting traffic destined to AS H and AS G using a BGP hijack and then dropping the corresponding connections (Fig. 6.1b). With SABRE, the network stays connected (Fig. 6.1c).

6.1 SABRE at a high-level

In this section, we provide an overview of SABRE's main operations and goals. Next, we describe the concrete attacker model SABRE aims at protecting against. Finally, we explain how SABRE's design allows it to live up to its goals.

SABRE is a relay network i.e., a set of nodes that contribute to the Bitcoin peer-to-peer network and aim at protecting clients from active routing attacks. To that end, SABRE allows every Bitcoin client to connect and provides them with an extra secure channel for learning and propagating the latest mined blocks.

To achieve this, SABRE needs to remains connected at all times, even under arbitrary BGP advertisements or extremely high load as we describe in Section 6.1.1. SABRE leverages two key insights: *(i)* smart positioning of the relay nodes to secure its internal connections and minimize the clients attack

surface (§ 6.1.2); and *(ii)* a hardware/software co-design for enabling relay nodes to endure almost arbitrary loads (§ 6.1.3).

6.1.1 Attacker model

We consider a slightly more powerful attacker model than we deem to be practical for splitting the network to half in Chapter 3. In particular, we assume that a partition attacker is aware of the SABRE and adapts her procedure accordingly.

We consider a single AS-level attacker whose goal is to partition the Bitcoin network into two disjoint components, S and N. For this purpose, the attacker first diverts the traffic destined to S or N, performing an interception attack using existing and more-specific prefixes (see § 1). After this step, the attacker *(i)* identifies the Bitcoin connections by inspecting the network and/or transport layer headers (i.e., by matching on IP addresses and/or TCP/UDP ports); and *(ii)* drops the connections bridging the partition. The explained attack is powerful because it can effectively partition the Bitcoin network [36]. The partitioning can potentially cause revenue losses and elicit double-spending [70]. Indeed, partitioning is an effective attack against any Blockchain system as it prevents nodes from communicating (§ 3.1).

We assume that the attacker knows: *(i)* the IP addresses of *all* SABRE nodes, along with *(ii)* the code running on them. As such, the attacker can hijack the prefixes hosting relay nodes and drop *all* traffic destined to them. Alternatively, the attacker can issue multiple requests to the relay nodes, effectively performing a (D)DoS attack.

Example We illustrate our attacker model using Figure 6.1a and 6.1b, which depict a simple AS-level topology composed of 9 ASes. ASB, ASD ASH, and ASG host Bitcoin clients that establish Bitcoin connections with each other (in blue). ASX is malicious and aims at disconnecting the nodes on the left side ($S = \{b1, d1, d2, d3\}$) from the others ($N = \{h1, g1, g2, g3\}$). For this purpose, ASX intercepts the Bitcoin connections directed to N by hijacking ASH and ASG prefixes. That way, ASX diverts all the connections from S to N and some others (e.g., the connection from $h1$ to $g1$). We depict the diverted connections in red in Figure 6.1b. Once on-path, the attacker drops the Bitcoin traffic *crossing* the partition and forwards the rest as usual. For instance, the attacker does *not* drop the connection between $h1$ and $g1$ and relays it from ASH to ASG as normal. Once the attacker launches the attack, Bitcoin clients in S can no longer communicate with Bitcoin clients in N; partitioning the Bitcoin network.

6.1.2 SABRE secure network design

We now explain SABRE relay locations' routing properties and how they protect relay-to-relay, client-to-relay, and relay-to-client connections from being disconnected by routing attacks. We describe an algorithm for systematically finding such locations in Section 6.2.

Protecting relay-to-relay connections A SABRE deployment is composed of relays hosted in $/24$ prefixes that belong to ASes that: *(i)* have no customer; *(ii)* peer directly; and *(iii)* form a k-connected graph. The combination of these three constraints protect relay-to-relay connections from routing attacks for three reasons.

First, these constraints prevent any attacker from diverting traffic between relays by advertising a more-specific prefix, effectively forcing her to advertise existing prefixes and thus compete with legitimate advertisements. *Second*, these constraints prevent any attacker from diverting relay traffic away from the ASes hosting relay nodes by advertising an economically preferred route. Indeed, the ASes hosting relays follow the advertisements of their direct peers to reach each other and have no customers (i.e., no better advertiser AS). As such, the number of malicious ASes which can advertise an equally-preferred route are limited to those that directly peer with the ASes hosting relays. Finally, these constraints make the chances for effective attackers to divert relay connections to exponentially decrease as the connectivity of the relay graph k increases. Indeed, BGP routers rely on an arbitrary tie-break to select among equally-preferred routes (e.g., by choosing routes learned from the lowest peer address [87]). Assuming that the attacker is equally likely to win this tie-breaking, she would only have a 3.1% (0.5^5) probability of disconnecting a 5-connected relay network. In Section 6.5, we show that well-connected relay networks are numerous.

Protecting client-to-relay connections While we can host relays in cherrypicked ASes, we cannot choose which ASes host Bitcoin clients. Concretely this means that AS-level adversaries hijacking relay prefixes can still divert connections originated by Bitcoin clients *to* the relays.

In SABRE, we protect client-to-relay connections by further restricting the locations in which we host relays to ASes whose BGP advertisements tend to be preferred by ASes hosting Bitcoin clients because of proximity or economic agreements. By doing so, we can lower the amount of traffic malicious ASes can divert, i.e., maximize SABRE's coverage. While individual relays locations are unlikely to protect many Bitcoin clients against all possible attackers, we show that a relatively small set of relays often can (§ 6.5). This design decision

is motivated by the observation that Bitcoin clients are concentrated in few hosting ASes [36].

Protecting relay-to-client connections Finally, an attacker might try to divert the traffic sourced from the relay network *to* the Bitcoin clients. For instance, an attacker could hijack the prefixes of Bitcoin clients and drop the relay connections by matching on any relay IP address. While this technique is more cumbersome for the attacker (there are way more clients than relays), it is nonetheless possible. SABRE relies on two techniques to obfuscate relay traffic. First, the relays can modify their source IP addresses when communicating with Bitcoin clients. This is possible as SABRE uses connectionless communications between the relays and the clients, enabling clients to accept packets with a different source IP than the one they send traffic to. Second, Bitcoin clients can connect to SABRE relays via a VPN/proxy service. Doing so would force the attacker to first find the mapping between the proxy IP and the corresponding Bitcoin client. As a result, SABRE renders this attack impractical. The attacker would need to perform a full inspection (beyond L4 headers) on a possibly huge volume of diverted traffic to distinguish the SABRE.

Example Using Fig. 6.1c, we now explain how a SABRE deployment of three relays, namely $r1$, $r2$ and $r3$, protects against routing attacks such as the one shown in Fig. 6.1b by securing intra-relay connectivity and maximizing coverage.

With respect to Fig. 6.1a, each Bitcoin client is now connected to at least one relay node in addition to maintaining regular Bitcoin connections. Here, nodes g_1, g_2, g_3 are connected to relay r_1 while node g_1 is also connected to node r_3. Hosted in ASes that peer directly, relay nodes protect their internal connectivity against ASX's hijacks. For instance, consider that ASX advertises the $/24$ prefix covering $r1$ to ASC. Since ASX is a provider of ASC, ASC discards the advertisement as it prefers to route traffic via a peer instead. At the same time, forming a 2-connected graph allows the relay network to sustain any single link cut. A link cut can be caused by a failure, an agreement violation or an unfiltered malicious advertisement from another direct peer.Observe that this would not hold if r_2 was not deployed. Finally, the exact positioning of relays is such that the paths towards them are more preferred over those of the attacker. As an illustration, ASX can divert the connection from ASG to ASD by advertising a more attractive path to ASG (as a peer) than the one it originally uses (a provider route, via ASF). Even so, ASX cannot divert the connection from ASG to ASB. Indeed, ASG will always prefer its customer path over any peer path.

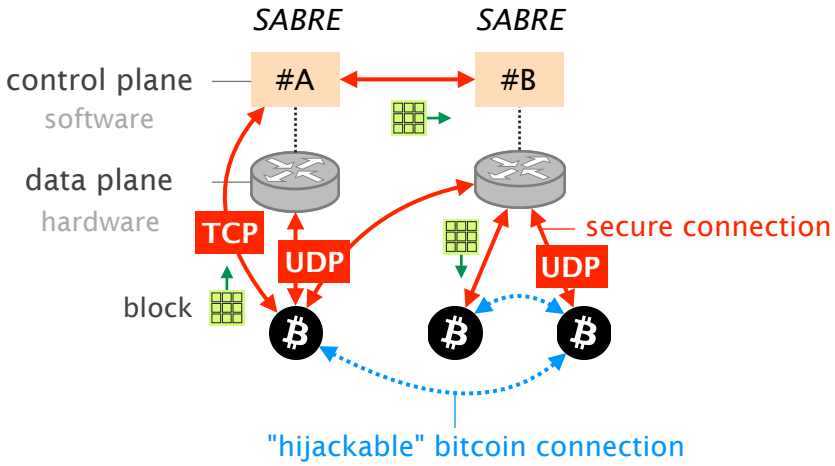


Figure 6.2 SABRE offloads most communication to the switch

6.1.3 SABRE resilient software/hardware node co-design

As a transparent and accessible relay network, SABRE nodes should be able to endure high load, either caused by legitimate Bitcoin clients or by malicious ones who try to exhaust their resources. To scale, SABRE nodes rely on a software/hardware co-design in which most of the operations are offloaded to programmable network switches (e.g., P4-enabled ones). We illustrate this deployment in Fig. 6.2 where two SABRE nodes are connected to each other and to some Bitcoin clients. One client talks directly to the controller via the switch, while the others talk only to the switch. Observe that a software-based implementation of the relay node would also protect Bitcoin against routing attacks, yet it will be more prone to (D)DoS attacks since it will have 2–3 orders of magnitude lower throughput [68] compared to a hardware-based one.[1]

SABRE's relay design is based on the observations that: *(i)* the content that needs to be cached in the relay node is predictable and small in size, consisting in the one or two blocks of 1MB that were most recently mined; and *(ii)* most of the relay operations are communication-heavy, consisting in propagating the latest known block to many clients and distinguishing the new ones. The former allows for effective caching, while the latter allows for partial hardware implementation in programmable network devices. This software/hardware co-design enables SABRE nodes to operate at Tbps and therefore sustain large (D)DoS attacks. Indeed, Barefoot Tofino programmable network devices can deal with as much as 6.5 Tbps of traffic in the backplane [21].

[1]We discuss the possibility of a software deployment of SABRE in Section 6.4.

While using programmable network devices enable high performance, it does not make it easy due to the lack of a broad instruction set and the strict limitations concerning memory and the number of operations per packet. We overcome these limitations with three techniques. First, our software/hardware design seamlessly blends in hardware and software operations, allowing to automatically escalate operations that cannot be done in the switch to a software component. In SABRE, only the validation of new blocks (which happens once every 10 minutes) needs to be escalated while all other requests are served by the hardware over a UDP-based protocol. Second, our implementation relies on optimized data structures which are both memory efficient and require a fixed number of operations per access. Third, we heavily precompute and cache values that would need to otherwise be computed on the switch (e.g., UDP checksums).

6.2 SABRE Secure network design

In this section, we first formally define the problem of selecting the ASes to host SABRE relays in (§ 6.2.1) so as to minimize the probability of a successful routing attack against Bitcoin. We then describe an algorithm for solving this problem (§ 6.2.2 and § 6.2.3).

6.2.1 Problem statement & challenges

The security provided by SABRE depends on: *(i)* how secure the intra-relay connectivity is, i.e., how many connections an AS-level adversary needs to hijack in order to disconnect the graph; and *(ii)* how much of the Bitcoin network is covered, i.e., how likely it is that an AS-level adversary will be able to prevent clients from connecting to *all* relay nodes.

We take both factors into consideration while constructing a SABRE network by first setting the desired level of intra-relay connectivity (e.g., 2-connectivity), and then finding the relay ASes that will maximize the Bitcoin coverage. Formally, we define our problem as follows:

Problem statement Let $G = (\mathcal{AS}, E)$ be the AS-level topology graph in which vertices ($\mathcal{AS}$) correspond to ASes and edges (E) to inter-AS links. Let also $\mathcal{B} \subseteq \mathcal{AS}$ be the subset of ASes that host Bitcoin clients and $\mathcal{R} \subseteq \mathcal{AS}$ be the subset of ASes that have no customers. Finally, let $G' = (\mathcal{R}, E')$ be the subgraph of G induced by $\mathcal{R}$ that contains the subset $E' \subseteq (\mathcal{R} \times \mathcal{R})$ of peer-to-peer inter-AS links. We define $\mathcal{A} = \mathcal{AS} \times \mathcal{B}$ as the set of all attack scenarios, namely all

pairs of ASes (x, v) in which AS x acts as AS-level adversary for AS v which hosts Bitcoin clients. Let $\mathcal{S} : \mathcal{R} \rightarrow \mathcal{A}$ be a function which, given a candidate relay, finds the subset $\alpha \subseteq \mathcal{A}$ of attack scenarios that this candidate relay protects against. Let furthermore $\mathcal{C} : \mathcal{P}(\mathcal{A}) \rightarrow \mathbb{R}$ be a function ($\mathcal{P}(\cdot)$ denotes the power set) which, given a set of attack scenarios $\alpha \subseteq \mathcal{A}$, quantifies their significance for the Bitcoin system by computing the number of Bitcoin clients hosted in victim ASes, i.e. $\mathcal{C}(\alpha) = \sum_{(x,v)\in\alpha} w_v$ where w_v is the number of Bitcoin clients affected by the attack scenarios in α.

Given a desired number of relays N and a desired inter-relay connectivity k, our problem is to maximize the number of attack scenarios Bitcoin clients are protected against. Formally, we aim at finding G'', a subgraph of G' induced by $\mathcal{R}'$, such that $\mathcal{R}' \subseteq \mathcal{R}$; $|R'| = N$; G'' is k-connected; and $\mathcal{C}\Big(\bigcup_{r_i\in R'} \mathcal{S}(r_i)\Big)$ is maximized.

Challenges Optimally solving the above problem is challenging for at least three reasons. First, the number of clients protected by any subset of relays R' depends on the *union* of the sets of the attack scenarios each relay $r \in R'$ protects against. As these are, in general, not disjoint, this problem reduces to the maximum coverage problem. Second, finding k-connected subgraphs in a random graph is difficult [48]. Third, in order to find the attack scenarios a relay protects against, one needs to predict the forwarding path each AS with Bitcoin clients will use to reach each candidate-relay considering *any* possible attacker.

We develop a heuristic to address the first two challenges (§ 6.2.2) and an algorithm for finding the possible attack scenarios for every attacker (§ 6.2.3).

6.2.2 Positioning SABRE relays

As described above, positioning relays optimally maps to solving a maximum coverage problem. Given the complexity, we rely on a greedy approach which is shown to be effectively optimal for the maximum coverage problem [54].

Our algorithm starts with an empty set R' and the set of candidate ASes $\mathcal{R}$ which satisfy the constraints listed in Section 6.1.2 and are also contained in at least one k-connected subgraph of at least N nodes, as only those can host one of the relay nodes of a k-connected network of N relays. It then iteratively adds relays to the set R' aiming at maximizing the number of covered attack scenarios while preserving k-connectivity for R'. This simple procedure runs in $\mathcal{O}(N)$ and works well in practice (Section 6.5).

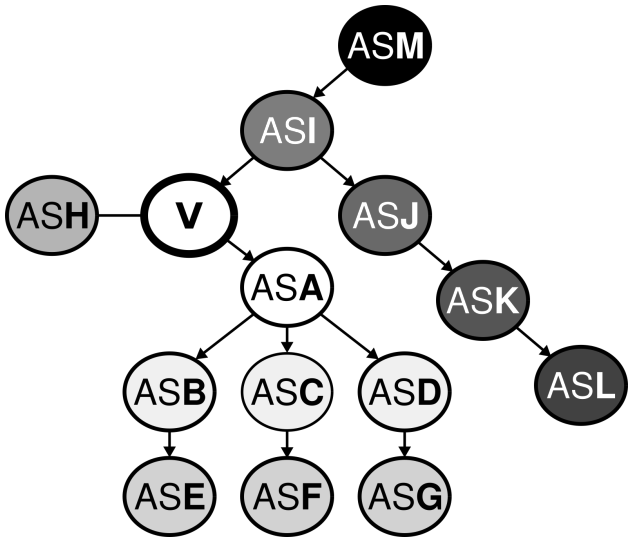


Figure 6.3 Shades illustrate ASV routing preference ranging from white (most preferred) to black (least preferred). Traffic from ASV to preferred AS is less likely to be hijacked.

In particular, in each round, we consider as candidates the set $\mathcal{R}_k \subseteq \mathcal{R} \setminus R'$ which are connected with at least $\min\{k, |R'|\}$ of the already-selected ASes in R'. Then we add the candidate $r \in \mathcal{R}_k$ that adds the maximum weighted coverage to R', i.e., the one with the maximum $\mathcal{C}\Big(\bigcup_{r_i \in (R' \cup r)} \mathcal{S}(r_i)\Big) - \mathcal{C}\Big(\bigcup_{r_i \in R'} \mathcal{S}(r_i)\Big)$ and we update R' accordingly, i.e, $R' := R' \cup \{r\}$. When we have selected all candidates, so that $|R'| = N$, we return R'.

We show in Section 6.5 that the resulting relay networks can readily protect between 80% to 98% of the existing Bitcoin clients (depending on the internal connectivity and number of deployed nodes) from 99% of the potential attackers. The exact algorithm for positioning the relay nodes can be found in *Algorithm* 3.

6.2.3 Calculating covered attack scenarios

Having explained how we can position SABRE relays based on the attack scenarios they cover, we now describe how we compute these scenarios for each relay, i.e., how we implement the function $\mathcal{S}$. More specifically, we describe how we predict, for each AS hosting Bitcoin clients and each AS-level adversary, whether the hosting AS will prefer the advertisements coming from the attacker AS over legitimate announcements coming from relay ASes.

Our algorithm is based on the observation that, to check whether an attacker AS (say ASM) can divert traffic from a victim AS (say ASV) to a relay AS (say ASR), we only need to compare the path from ASV to ASR and from ASV to ASM. If the path to ASM is more preferred, then ASM can successfully hijack traffic from an ASV to ASR. Path preference is dictated by the business relationships established between ASes together with the path length: customers are preferred over peers, peers over providers, and shorter paths over longer ones.

As an illustration, Figure 6.3 illustrates an AS topology in which arrows indicate business relationships: providers are drawn above their customer (ASV is the provider of ASA), while peers are drawn alongside each other (ASH and ASV are peers). The different shades indicate which advertisements ASV prefers, ranging from white (most preferred) to black (least preferred). ASV prefers advertisements learned from ASA (its customer) over advertisements learned from ASH (its peer) or from ASI (its provider). Likewise, ASV prefers advertisements from ASE over the ones originated by ASD. While both are learned via ASA (its customer), ASD is closer to ASV (2 hops) than ASE (3 hops). Intuitively, Figure 6.3 depicts for any two ASes ASX and ASY whether ASV would prefer ASX's advertisements over ASY's, should both advertise the same prefix. For instance, if a relay is hosted in ASH (ASV peer), all the ASes that ASV can reach via customer links (ASA–ASG) are possible attackers. Similarly, if the relay is hosted in ASM then any other ASes (ASA–ASL) can divert the corresponding traffic from ASV.

We describe the algorithm we use to compare the BGP preference of two paths with a common start in the Algorithm 4.

6.3 SABRE Resilient relay node design

We now explain the software/hardware co-design of a SABRE node (§ 6.3.1) and its operations (§ 6.3.2), which ensure that the node's resources are not maliciously exhausted and that benign clients are not denied service.

6.3.1 Hardware/software co-design

Figure 6.4 illustrates SABRE's software/hardware co-design. It is composed of a programmable switch connected to a modified Bitcoin client, which acts as a controller.

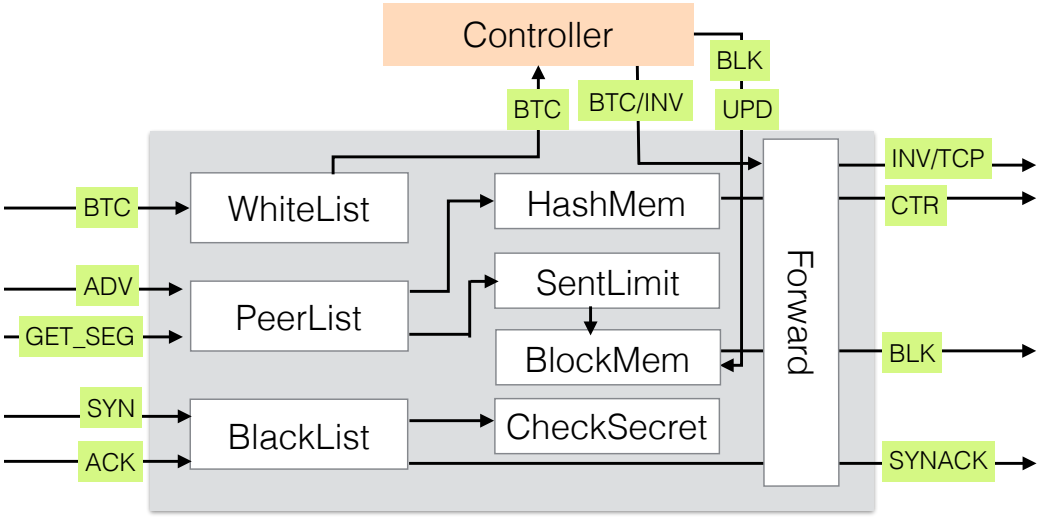


Figure 6.4 The switch intercepts all incoming traffic, answers to all UDP requests and redirects TCP traffic of whitelisted clients to the controller. The switch contains the latest mined Block in *BlockMem* and multiple components to track the connected and banned clients (e.g.White/Black List, Connected) and detect attacks (e.g. CheckSecret, SentLimit)

The switch is responsible for: *(i)* serving client connections; *(ii)* protecting the controller only from malicious clients; *(iii)* propagating blocks; and *(iv)* distinguishing new blocks from old ones. In contrast, the controller is responsible for validating new blocks, advertising them to the connected clients and updating the switch memory accordingly.

Bitcoin clients establish UDP connections with the switch and (rarely) regular Bitcoin connections (over TCP) with the controller. Switches only allow approved Bitcoin clients to establish connections with the controller. As most clients "consume" blocks rather than producing them, we expect most clients to only interact with SABRE's hardware component.

SABRE defines a UDP-based protocol to facilitate communication between the Bitcoin clients and the switch as well as between the switch and the controller. The protocol is composed of 8 messages. Five of them are exchanged between the switch and the clients: SYN, SYN/ACK, ACK, GET_SEQ, and ADV. The three remaining are sent between the switch and the controller: NCONN, UPD, and BLK.

Similarly to TCP, SYN, SYN/ACK, ACK are used to prevent spoofing attacks. GET_SEQ, BLK and ADV relate to block management. Specifically, GET_SEQ enables a client to request a particular segment of a block which is sent as a

BLK, while ADV enables a client to advertise a newly mined block to the relay. The switch sends a NCONN to notify the controller of new connections. The controller sends an UPD followed by a BLK message to update the switch with the latest block.

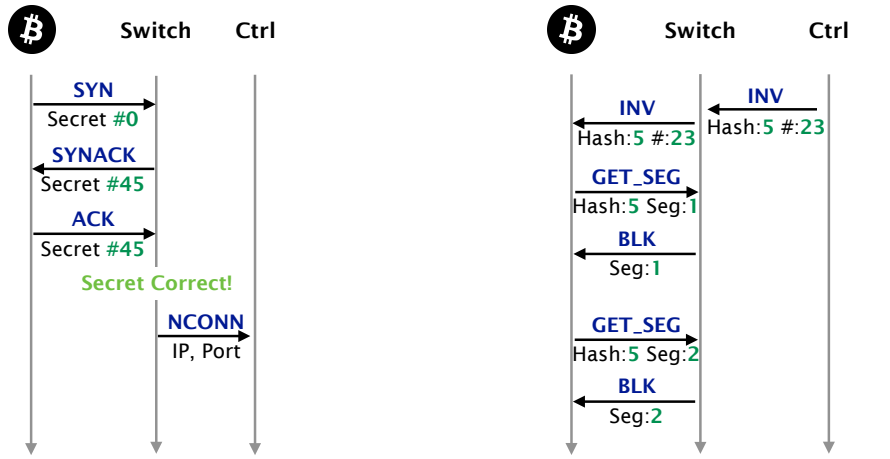
The switch maintains three data structures to manage client connections and track down anomalies: *PeerList*, *Whitelist*, and *Blacklist*. *PeerList* contains information about connected clients, i.e., those who successfully completed the three-way handshake. Similarly, *Whitelist* stores clients that are allowed to communicate with the controller directly. *Blacklist* contains clients that have misused the relay and are banned. The switch also maintains one data structure to store the latest block (s): *BlockMem*. *BlockMem* is composed of indexed equal-sized segments of a block together with precomputed checksums for each segment. This data structure allows the switch to timely reply with the requested segment avoiding additional computations. Moreover, the switch contains two components devoted to anomaly detection: *SentLimit* and *CheckSecret*. *SentLimit* detects clients that requested a block too many times. *CheckSecret* verifies during the handshake that a client is using its true IP. Finally, the switch also maintains one data structure for checking whether an advertised hash is new or known: *Memhash*.

In the following, we describe the different operations performed by the relay and how each of them modifies each of the data structures. In Section 6.6, we show that our design can sustain 1M malicious and 100k benign client connections with less than 5 MB of memory in the switch. This memory footprint is only a fraction of the memory offered by programmable switches today [69], allowing the switch to be used for other applications.

6.3.2 Relay operations

We now describe SABRE relay operations. The client and controller are extended versions of the default Bitcoin client and the switch is implemented in P4 [44]. Our protocol defines four operations: *(i)* how regular Bitcoin clients connect to a relay node; *(ii)* how a relay node propagates blocks back to them; *(iii)* how a relay node receives and validates blocks transmitted by the clients; and *(iv)* how the controller updates the switch memory upon receiving a new valid block.

Managing client connections In order to avoid spoofing attacks, Bitcoin clients initialize connections to relay nodes using a three-way handshake as shown in Figure 6.5a. As for a normal TCP connection, the client first sends a SYN packet. Upon receiving the SYN, the switch echoes back a secret value calculated using



(a) Establishing connection **(b)** Transmitting Block

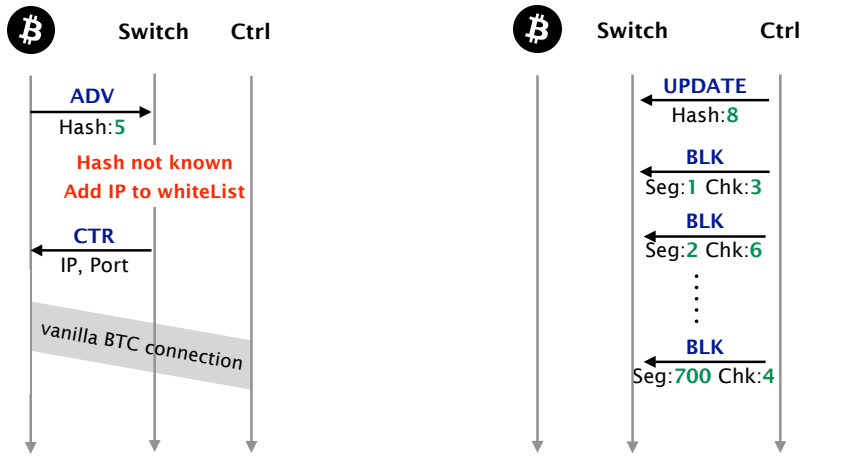
Figure 6.5 (a) BTC client establishes a connection with the switch using a 3-way handshake. (b) Relay advertises a new block INV via the switch and transmits it using multiple BLK messages after client requests using GET_SEG messages.

the client's IP address and UDP port in a SYN/ACK packet. The client then includes this secret value in the final (ACK) packet as proof that it owns the source IP address that it is using.

Upon successfully completing the handshake, the switch adds an entry for the client's IP and port number in the *PeerList* and notifies the controller via a NCONN message. The *PeerList* is implemented as a Bloom filter (BF) for memory efficiency. Doing so, the switch verifies that an incoming packet belongs to an established connection and drops it otherwise. As BFs do not support listing all inserted items, the controller stores the connections for future use (e.g., advertising new blocks and updating the *PeerList*).

Learning new blocks Relay nodes need to learn new blocks that are mined. New blocks are transmitted to the relays from regular clients. Being a network device with limited computational capabilities, the switch is unable to validate blocks. Thus, advertised blocks need to be transmitted to the controller after they have been filtered by the switch.

As illustrated in Fig. 6.6a, the node advertises a block by its hash to the switch using an ADV message. The switch checks whether the hash is already known using the HashMem. If the hash is not known, then the switch asks the client to connect to the controller with a CTR message and stores its IP in the *Whitelist*. If the transmitted block is legitimate, the client's IP will stay in the whitelist for four days. The client connects to the controller as if it was a regular Bitcoin



(a) Receiving block **(b)** Updating switch

Figure 6.6 (a) BTC client advertises a new block identified as unknown by the switch. The client gets white-listed, allowing it to connect directly to the controller. (b) If the received block is valid, the controller updates the switch using an UPDATE message carrying the block's hash followed by a BLK messages carrying the data.

client, while the switch forwards the TCP traffic to the controller. The switch only allows packets from white-listed clients to reach the controller. Observe that a malicious miner cannot monopolize or overload the controller with its connections as even a pool with 30% of the hash power cannot keep more than 172 whitelisted clients at any given moment.[2]

Even so, a malicious miner might still try to engineer block races by flooding the relay node with multiple blocks simultaneously, which will need to be validated by the controller. To shield against this attack, the switch keeps the number of active nodes that are white-listed. When this number exceeds a predefined threshold (set based on the controller's hardware capabilities), the switch will stop whitelisting new clients. In this case, the controller receives blocks from the nodes that are already whitelisted. Indeed, these nodes are diverse enough, with respect to mining power origin, to keep the relay up-to-date, thanks to the expiry mechanism in the *Whitelist*. For instance, any pool with at least 0.17% of mining power can keep at least one node in the *Whitelist* forever. In essence, the

[2]Every day, 144 Blocks are mined (on average). For each block at most one node is whitelisted (the one that is not already whitelisted and advertised the block first)

switch implements a simple-yet-efficient reputation-based access-list to protect the controller from Sybil attacks.

Updating the switch with a new block If a newly-transmitted block is valid, the controller updates the switch's memory with a new mapping of segment IDs to data segment that corresponds to a particular block hash. The switch can then transmit the segments to the clients upon requests. Observe though that the switch sends data to a UDP socket. Thus, the IP and UDP checksums need to be correct for the packet to be accepted. The UDP checksum is calculated using a pseudo-header and the one's complement sum of the payload split into 16 bits segments. Since computing this in the switch would result in repetitive and unnecessary computations, we precompute the one's complement sum of the block segment and cache it together with the segment itself. Using this value the switch needs only to add the header parts that are different per client to calculate the checksum.

Figure 6.6b illustrates the sequence of packets the controller sends to update the switch. Initially, it sends an UPD message containing the new hash. This first message tells the switch to prepare its state for the new block. The next messages are sent to transmit each of the segments of the block as well as the precomputed one's complement sum.

Propagating a newly-learned block The relay node advertises new blocks to all its connected clients, who can then request a block segment-by-segment. Blocks are transmitted in multiple individual segments for three reasons: *(i)* to allow clients to request lost segments independently; *(ii)* to avoid loops in the data plane, which would otherwise be necessary as the block does not fit in one packet; and *(iii)* to protect against amplification attacks.

As illustrated in Figure 6.5b, the controller sends an INV message, which is forwarded by the switch. This INV message contains the hash of the new block as well as the number of segments it is composed of. In the example, the relay advertises hash $\#5$, which is composed of 23 segments. If the Bitcoin client is unaware of the advertised block, it requests it using a GET_SEG message containing the hash of the block and each of the 23 segment IDs. In the example, the client first requests the segment of $ID{:}1$ of the block with hash $\#5$ then the segment of $ID{:}2$ and so on. If either the GET_SEG or the SEG is lost the client will simply request the corresponding segment again. As a protection mechanism, the switch bans clients that request a block multiple times. To that end, all requests traverse a heavy-hitter detector, namely *SentLimit*. For efficiently implementing this component, one can reuse [91] which can operate with just 80KB of memory.

6.4 Deployability & incentives

In this section, we show that SABRE is both practical and partially deployable. While a full deployment of SABRE can protect Bitcoin as a whole, partially deploying SABRE is less expensive and offers gains to early adopters (even individuals).

6.4.1 Full deployment

A complete deployment of SABRE requires: *(i)* hosting relays in particular ASes; *(ii)* equipping relay locations with specialized hardware; and *(iii)* incentivizing a third party to build and maintain the infrastructure. In the following, we explain why each of those requirements is practical.

First, we observe that a large number of ASes that can host SABRE relays are cloud providers, CDNs, IXPs, large ISPs, or Software-as-a-Service (SaaS) providers. This should come as no surprise as such ASes are actively trying to establish as many peering connections as possible to improve their services. Deploying SABRE nodes in such ASes is practical as they already sell online services or are research-friendly (IXPs [63, 62]). Moreover, even if some eligible ASes do not consent to host SABRE nodes, the effectiveness of SABRE will not be significantly affected as: *(i)* SABRE only requires few nodes to be useful (as little as 6 ASes, see § 6.5); and *(ii)* there are more than 2000 possible locations for hosting ASes. In short, no candidate AS is irreplaceable.

Second, we argue that cloud providers could start renting out hardware-accelerated computing instances with programmable network data planes. Actually, some cloud providers already allow clients to rent advanced hardware resources. For instance, Amazon EC2 offers the possibility to connect computing instances with field-programmable gate arrays [20]. That said, a pure software-based implementation of SABRE would still protect the Bitcoin network from routing attacks, leaving DDoS protection to the default mechanism operated by each AS. As described above, such a software-based implementation could be readily deployed as it only requires the possibility to host virtual machines in candidate ASes.

Third, we argue that the possible monetary losses induced by routing attacks [36] create business incentives for one or more entities to deploy and maintain relay nodes. We observe that such incentives are similar to the ones behind existing relay networks such as FIBRE [8] and Falcon [7].

Observe also that deploying SABRE does not need to be approved by the community as a whole. Indeed, a regular client can connect to a SABRE node via a single lightweight UDP connection by independently upgrading its code. Thus, the notoriously slow-moving Bitcoin community cannot be an obstacle to SABRE's deployment.

6.4.2 Partial deployment

While feasible, fully deploying SABRE is time-consuming and requires multiple parties to collaborate and possibly share costs. Luckily, SABRE can also be partially deployed, e.g. by a mining pool that wishes to protect itself from routing attacks or by existing relay networks that wish to improve their poor 6.5.2 protection against routing attacks.

By deploying SABRE at a low-budget, a single mining pool can secure the propagation of its own blocks and the reception of new mined ones, even while the Bitcoin peer-peer-network is under a severe routing attack. As an illustration, such a deployment of 6 SABRE nodes would only cost 500$/month (considering the current AWS pricing policy [23]). This cost is: (i) well within the financial capabilities of a single mining pool; (ii) entirely justified given the possible losses that a pool could incur upon a successful routing attack (e.g. an orphan block is a 150K loss [25]).

Note that the above deployment does not require special hardware to scale as the pool is only interested in serving its own Bitcoin clients (i.e. its gateways) rather than any possible Bitcoin client that might wish to benefit from the SABRE.

Of course, as multiple pools start to build SABRE networks, they can collaborate and share costs, henceforth incentivizing the deployment of larger SABRE networks (possibly using hardware-accelerated instances which are also already sold by cloud providers [25]) that could protect more clients.

At the same time, existing relay networks such as FIBRE and Falcon can also have significant gains by partially deploying SABRE as its relay location algorithm is orthogonal to their approaches. To do so, existing relays need to independently relocate their servers accordingly to SABRE's network design.

6.5 Network architecture evaluation

In this section, we evaluate SABRE's efficiency in protecting Bitcoin against routing attacks. Specifically, we answer the following questions: How effective is SABRE in preventing routing attacks targeted against the entire network and individual clients? How does this effectiveness change with the size and the connectivity of the SABRE network? How does SABRE stand out against other relay networks and known countermeasures?

We found that even a small deployment of 6 single-connected SABRE nodes can prevent 94% of ASes in the Internet from isolating more than 10% of the Bitcoin clients; while larger deployments of 30 relays that are 5-connected can prevent more than 99% of the ASes from isolating more than 20% of Bitcoin clients. In addition, we show that existing relay networks, like Falcon [7] and FIBRE [8], offer *no* protection against routing attacks. Finally, we show that SABRE provides security level on-par with hosting all clients in $/24$, an effective but clearly impractical countermeasure.

We start the section by describing our methodology before presenting our results in detail.

Methodology Our evaluation relies on a joint dataset combining routing and Bitcoin information processed as describe in Section 2. We use the IPs of existing relay nodes from [8, 7], both collected in May 2018. Notably, we assume that the attacker's advertisements are systematically picked at the tie-breaking state of the BGP decision process (the worst-case for SABRE).

6.5.1 SABRE security efficiency

SABRE protects against network-wide partitions To evaluate how effective SABRE is against adversaries that wish to partition the Bitcoin network, we quantify how likely it is for a random adversary to be able to disconnect multiple clients from the relay network. The fraction of clients a particular AS can disconnect from the relays is the maximum set of Bitcoin clients she can isolate, as Bitcoin nodes connected to the relay network cannot be partitioned.

Fig. 6.7 illustrates how protected the Bitcoin network is depending on the size N and internal connectivity k of the SABRE network. The graph shows, for each given fraction y of Bitcoin nodes, the percentage of ASes that would be able to independently disconnect it from SABRE . For $N = 20, k = 1$, less than 3% of ASes are able to prevent a considerable fraction of Bitcoin clients (15%) from

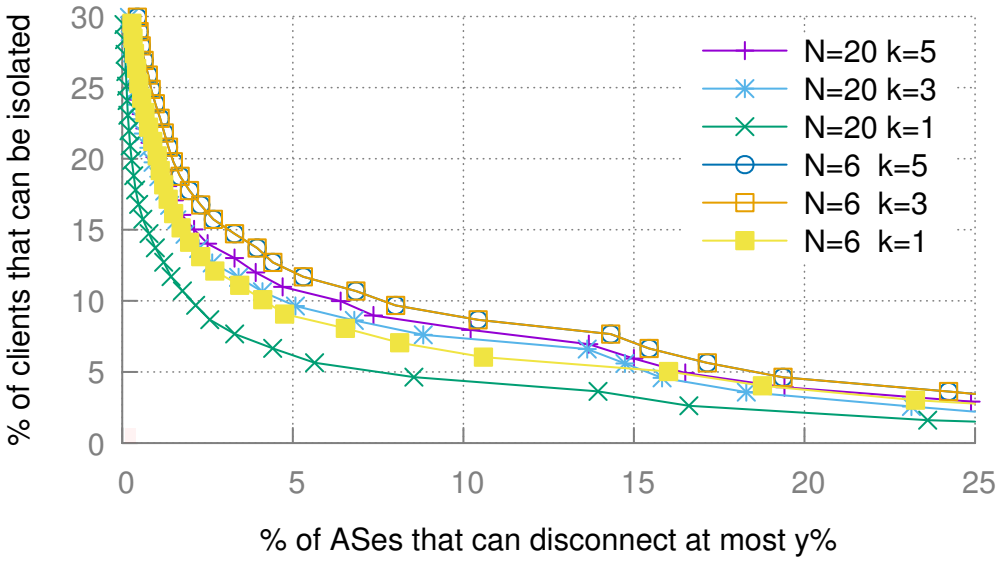


Figure 6.7 Less than 2.5% of ASes are able to disconnect more than 15% of clients (N: the number of deployed relays; k: relay-graph connectivity; Tie breaks in favor of the attacker).

connecting to the relay network. In contrast, more than 90% of the clients can be isolated by *any* AS in the current network [36].

The mapping between the number of possible attackers and the partition sizes varies with the size and connectivity of SABRE. In particular, increasing the number of deployed nodes decreases the chances that adversaries can divert traffic successfully. On the other hand, decreasing the intra-connectivity requirements (i.e., the value of k) allows our algorithm (§ 6.2) to select from a larger set of relays and thus to form a more effective SABRE. This creates an interesting trade-off between how secure the intra-relay connectivity is and how well the relay nodes cover the existing Bitcoin network. For example, while a SABRE of 6 relays that are connected in full-mesh (5-connected graph) is extremely hard to partition, as the AS-level adversary would need to divert 5 peer-to-peer links, it enables more AS-level adversaries to disconnect a larger part of Bitcoin clients from SABRE. For example, 3% of ASes can potentially create a partition including 22% of Bitcoin nodes. In contrast, a 1-connected SABRE allows fewer attackers to perform severe attacks—only 1% of ASes could create a 12% partition—but the relay network itself can be partitioned by a single link failure or a successful hijack from a direct peer.

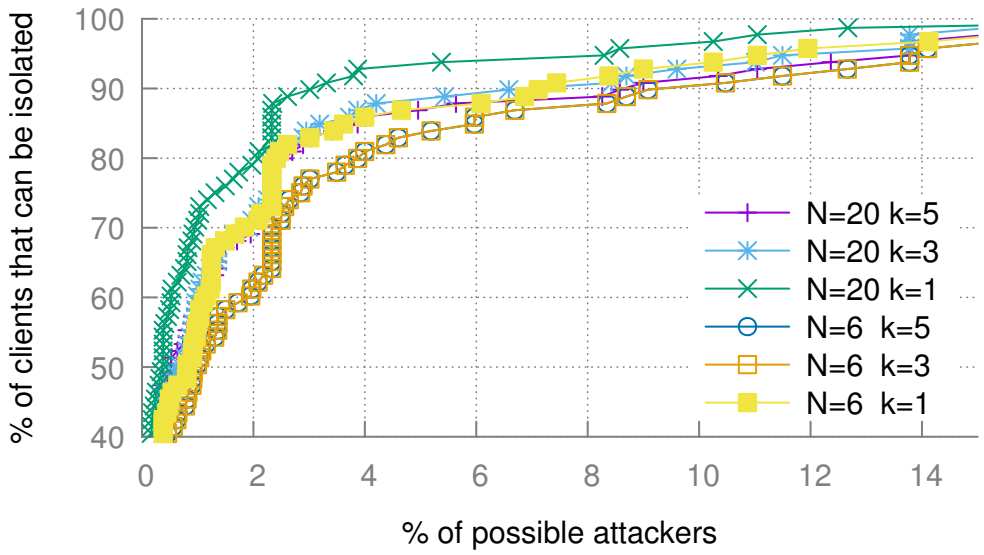


Figure 6.8 85% of the clients are protected against 96% of possible attackers (Tie breaks in favor of the attacker)

SABRE protects most individual clients To evaluate how effectively SABRE protects individual clients, we look at how likely it is for Bitcoin clients to be prevented by a random AS-level adversary from reaching *all* relay nodes.

Fig. 6.9 shows, for each given percentage of ASes in the Internet, the percentage of Bitcoin clients could be attacked and disconnected from SABRE by this percentage of ASes. We see that 80% of the clients are protected from 96% of the AS-level adversaries even with a SABRE network of only 6 nodes that are 5-connected. There is again a trade-off between secure intra-connectivity and the coverage of Bitcoin clients. For example, a SABRE of 6 1-connected nodes protects 90% of Bitcoin clients from 92.5% of ASes, while a fully connected 6-node SABRE protects from only 89.5% of ASes. Interestingly, increasing connectivity from $k = 3$ to $k = 5$ does not decrease the protected clients significantly while making disconnecting the relay network almost impossible.

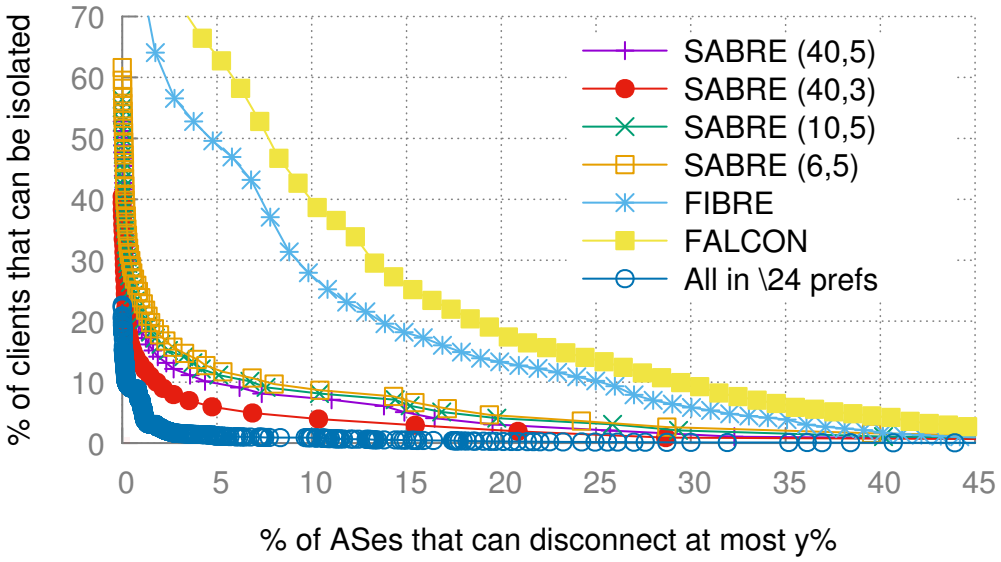


Figure 6.9 SABRE is far more secure than deployed relays and very close to the unemployable alternative countermeasure of hosting all clients in $/24$. (Tie breaks in favor of the attacker)

6.5.2 SABRE efficiency compared to existing relay networks

We compare SABRE to FIBRE [8] and Falcon [7] with respect to their effectiveness against routing attacks. We found that SABRE outperforms both, for three key reasons.

Existing relays are vulnerable to longer-prefix hijacks *All* relay nodes of both FIBRE and Falcon are hosted in prefixes that are shorter than $/24$. As such, *any* AS-level adversary can cut connections among relays as well as from the Bitcoin clients only by hijacking 6 more-specific prefixes for FIBRE and 10 for Falcon.

Existing relay networks are poorly connected Even if these relay networks were hosted in /24 prefixes, our analysis revealed that their connections could still be diverted by same-prefix advertisements. In particular, we found that FIBRE relays would be disconnected by any of 652 ASes, and Falcon by any of 3 ASes even if $/24$ prefixes were used.

Existing relays provide bad coverage Even ignoring their poor relay-to-relay connectivity and again assuming that these relay networks were hosted in $/24$ prefixes, their client-to-relay connections would still have been more vulnerable

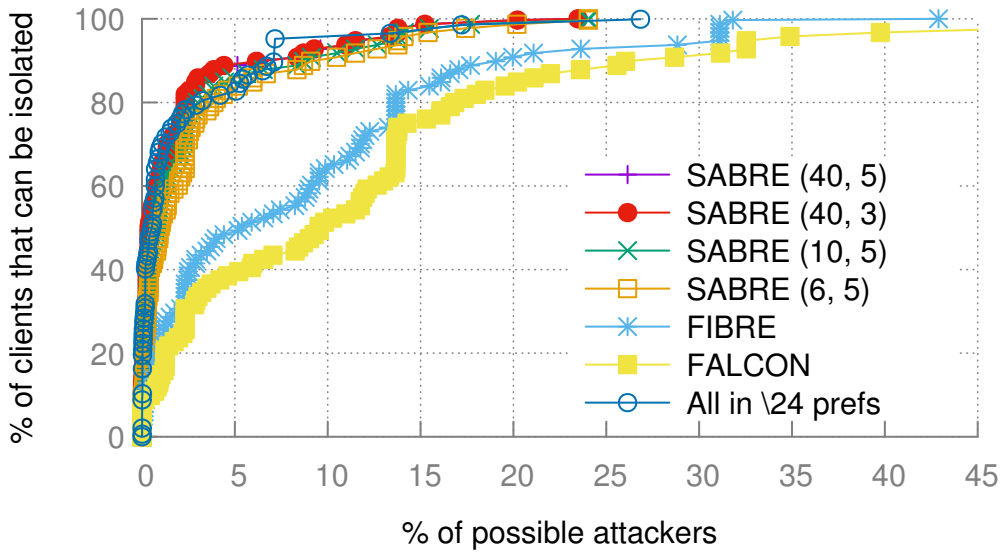


Figure 6.10 Falcon does not protect many clients as it is centralized to only two ASes. SABRE performs on-par with hosting all clients in $/24$ while being deployable (Tie breaks in favor of the attacker)

than those of SABRE allowing for more network-wide and targeted attacks. We compare existing relay networks with SABRE with respect to how well they protect against routing attacks using the same graphs as in Section 6.5.1. In particular, Fig. 6.9 shows the percentage of ASes that are able to independently isolate a fraction of the Bitcoin network as a function of this fraction while Fig. 6.10 shows the cumulative percentage of clients as a function of the number of AS-level adversaries that could disconnect them from all relay nodes. While FIBRE is slightly better than Falcon, SABRE outperforms both.

6.5.3 SABRE efficiency compared to hosting all clients in /24s

We now compare SABRE to the most effective countermeasure against routing attacks: hosting *all* Bitcoin clients in /24 prefixes [36]. While effective, this countermeasure is also highly impractical as it requires ISP cooperation in addition to an increase in the size of the routing tables Internet-wide. We found that SABRE offers a comparable level of protection against network-wide and targeted attacks while being easily deployable.

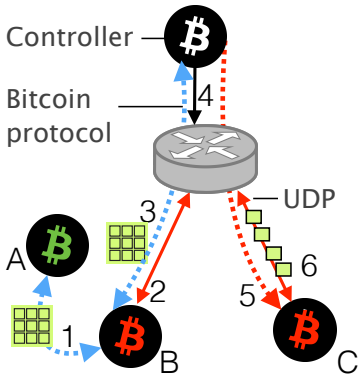
The comparison between the two approaches is not straightforward as SABRE protects the network even if the attacker has already partitioned the Bitcoin Peer-to-Peer network while the other approach (hosting clients in $/24$) aims at securing the Peer-to-Peer network itself. We compare them against the same metrics we used previously in Section 6.5.1, namely offered protection against partition attacks and effectiveness in protecting individual nodes. In the following, we describe how we calculated those metrics for the all-$/24$ approach before presenting our results.

First, we estimate the size of different partitions and the number of AS-level adversaries that could achieve those assuming all clients are hosted in $/24$ prefixes. In particular, we find the AS-level adversaries that would be able to isolate a considerable fraction of Bitcoin clients using same-prefix advertisements only. To that end, we run a search on the AS-level topology graph staring from each AS with Bitcoin clients X and following order of descending path preference (as described in Section 6.2.3), namely more economically preferred paths for X are visited first. All ASes that are traversed by the search before another AS with Bitcoin clients are able to isolate X from the Bitcoin network. Indeed, this calculation gives only a lower bound with respect to the possible partitions, i.e., hosting all clients in $/24$ prefixes might offer less security than what we computed. Finally, we compare our findings to SABRE's. Our results are summarized in Fig. 6.9. Indeed, hosting all clients in $/24$ prefixes would secure the Bitcoin Network better than SABRE, as partitions larger than 20% would be possible for only 0.016% of ASes.

Second, in order to calculate how many attackers can successfully isolate individual Bitcoin clients, assuming that all clients are hosted in $/24$ prefixes, we looked at the ASes that are able to divert traffic from each of those clients to all others in the network. We compare our findings to SABRE's. The results are included in Fig. 6.10. The two approaches show similar protection levels, with SABRE being slightly better at times. This is because SABRE can place relays in any AS in the Internet, while the alternative countermeasure is limited to the actual distribution of Bitcoin clients.

6.6 Software/hardware co-design feasibility

We validated the feasibility of our co-design by testing it in practice using regular and modified Bitcoin clients. In this section, we showcase that: *(i)* a programmable switch can seamlessly talk to a Bitcoin client without any software



	Time	Description
1	0	Node B gets block from node A
2	69ms	Node B sends ADV to switch
3	14ms	Node B connects to Controller
4	7s	Controller updates the switch
5	1.6ms	Controller advertises block to C
6	7.22s	Node C gets block in segments

Figure 6.11 A block can be successfully transmitted from node A to node C via the SABRE after it has been validated by the controller.

interaction; and that *(ii)* the data-plane memory footprint is low compared to the on-chip memory available in today's programmable switches.

Implementation and testbed Both the controller and the clients are implemented as extensions of the default Bitcoin client version 0.16. The former containing ~650 added or modified lines of C++ code and the latter ~680 lines. The switch is implemented in ~900 lines of P4 code. Our prototype runs on Mininet [75] and uses the publicly available P4 behavioral model (BMV2) [26] to emulate the switch. Our testbed (see Fig. 6.11) is composed of three clients A, B, C along with a relay node consisting of a switch and a controller. Nodes B, C (shown in red) are modified and are connected to SABRE, while node A (shown in green) is an unmodified Bitcoin client.

Timing We walk through the life of a block that was mined in the Bitcoin network and sent by the unmodified client A to B. The latter will advertise the new block to the switch, which will allow node B to connect directly to the controller and transmit it. The controller will then update the memory of the switch and will advertise the block to the connected peers (e.g., C). Next, node C will request and receive the block in segments. The main steps of this procedure are listed in Fig. 6.11 which describes each step and the time it required in our prototype implementation. The most time-consuming operations are updating the switch and transmitting the block, taking ~7s each. These high delays are due to the fact that we relied on a software-based P4 switch. In practice, the only actual bottleneck in a hardware implementation would be the uplink of the relay nodes.

Memory requirements We analytically calculated the memory required for each of the components in the switch, taking into consideration the expected

Component	*Items*	*False Positive*	*Memory*
BlackList:	1000000	0.001	1.80MB
WhiteList:	100	0.0001	239.75B
HashMem:	518823	0.0001	1.24MB
PeerList:	100000	0.0001	479.25K
blockMem:	1	-	1.0MB

Table 6.1 The memory used in the P4 switch is always <5MB

load. Table 6.1 summarizes our results. It contains the name of the component and its capacity, i.e., the number of elements that can be added such that the false positive rate listed in the third column is not exceeded. We found that the cumulative memory needed does not exceed 5MB, which is well within the limitations of today's programmable switches. The most memory-demanding component is the *Blacklist* for which we budget 1 million entries. This is necessary to allow for mitigating DDoS attacks. In contrast, the components devoted to benign operations are less memory-demanding since the number of legitimate clients is significantly less. For instance, we only reserve space for $100k$ clients in the *PeerList* and 100 for the *Whitelist*; both require less than 1MB. Observe that Bloom filters for regular clients have a lower false-positive rate than the *Blacklist*. This enables to serve already connected clients even if the switch is under such an extreme DDoS attack that the *Blacklist* is flooded. We do not list the requirement of the *SentLimit* component as they are negligible [91]. Finally, the memory needed for storing the latest block as well as for keeping all known hashes takes about 1MB each.

Algorithm 3 Algorithm to find the best set of relays to connect.

```
begin
    function LocateRelays(C, C_scens, N, k) %C_scens scenarios that
    each relay protects against
    %N number of relays to deploy
    %k desired connectivity
        R ← {}%Relays to be deployed
        R_scens ← {}%Scenarios R protects against
    while R.length < N do
            Cs ← {c : c ∈ C \ R s.t.G[R ∪ c]is k-connected}
            best_r ← FindNext (Cs,C_scens,R_scens)
            R.add(best_r)
        return R
        function FindNext(Cs, C_scens, R_scens)
            best_r ← None
            best_scens ← {}
            best_effect ← MAX
    for r in Cs do
                tmp_scens ← R_scens ∪ C_scens[r] if R_scens.effect <
    best_effect then
                    best_scens ← tmp_scens
                    best_effect ← R_scens.effect
                    best_r ← r
            return best_r
```

Algorithm 4 Compare two paths based on preference.

```
begin
    function MorePreferred(pathA, pathB)
        typeSeqA ← path_type(pathA)
        typeSeqB ← path_type(pathB)
    while pathA & pathB & hopA.pick==hopB.pick do
            hopA ← pathA.pop()
            hopB ← pathB.pop()
            typeA ← typeSeqB.pop()
            typeB ← typeSeqB.pop()
    if typeA ≠ typeB then
        if (typeA, typeB) == (customer, peer) then
                return 0
        if (typeA, typeB) == (customer, provider) then
                return 0
        if (typeA, typeB) == (peer, provider) then
                return 0
        if (typeA, typeB) == (peer, customer) then
                return 1
        if (typeA, typeB) == (provider, customer) then
                return 0
        if (typeA, typeB) == (provider, peer) then
                return 1
    else
        if len(pathA) = len(pathB) then
                return 1 %In case of a tie, we prefer path B.
        else
                return argmin(len(pathA), len(pathB))
    end function
```

7

Defenses against passive attacks

In this chapter, we present a set of countermeasures against passive routing attacks. We distinguish these countermeasures into two categories: protocol-based techniques and deployment strategies. The former category comprises countermeasures that require support from all clients to be effective (§ 7.1). The latter category comprises countermeasures that can benefit individual nodes but are not practical network-wide as they require third-party support or/and do not scale (§ 7.2).

7.1 Protocol-based techniques

Encrypting traffic One of the most critical enablers of passive attacks in Bitcoin is that traffic is routed over the Internet unencrypted. Thus, encrypting Bitcoin's traffic would make the currency less vulnerable to passive attacks as it would prevent the attacker from modifying and observing the content of the exchanged messages. However, While encryption will make passive AS-level attacks harder, this alone is not adequate mitigation. A passive AS-level adversary would still be able to observe metadata and/or drop connections. In fact, we have already presented an attack that generalizes to encrypted traffic: the perimeter attacker can deanonymize Ethereum clients even though Ethereum traffic is encrypted.

This is possible because the headers of all exchanged packets travel in plain text allowing the attacker to infer a client's peers i.e., useful metadata information

Obfuscating traffic Since encryption (i.e., obfuscation of the message content) is not an adequate countermeasure against AS-level attackers, we should consider obfuscating other differentiators that make cryptocurrency traffic easily identifiable. For instance, both the Bitcoin and the Ethereum protocol use a well-known port on which the attacker can filter to distinguish the connections of interest. Hiding all such differentiators of cryptocurrency traffic would make passive attacks either easily detectable or unrealistically expensive. Indeed, the attacker would need to either act on *all* the traffic she intercepts (e.g., by delaying or dropping traffic to a certain destination IP) or invest in sophisticated hardware to distinguish the traffic (e.g., deep packet inspection).

The first step towards obfuscating traffic is the use of distinct control and data channels. The two ends could negotiate a set of random TCP ports upon connecting to each other using the well-known port. Next, the two ends could use the random ports to establish the actual TCP connection, on which they will exchange data. This would force the AS-level adversary to look at *all the traffic*, which would be too costly.

A simpler (but poorer) solution would be for clients to use a randomized TCP port (encoded in the clear with the ADDR message) as it will force the AS-level adversary to maintain state to keep track of these ports. Although a node can already run on a non-default port, such a node will receive fewer incoming connections (if any) as the default client strongly prefers peers that run on the default port.

Select Bitcoin peers while taking routing into account Bitcoin nodes establish outgoing connections randomly while avoiding initializing multiple connections to the same IP prefix. While randomness is important to avoid biased decisions at the application level, it also results in a few ASes intercepting a significant percentage of Bitcoin connections due to the Internet's and Bitcoin's routing centralization (see Chapter 2). To avoid this, Bitcoin nodes should establish a few *extra* connections taking the underlying Internet structure into consideration. For this, nodes could either issue a `traceroute` to each of their peers and analyze how often the same AS appears in the path or, alternatively, tap into the BGP feed of their network and select their peers based on the AS-path. In both cases, if the same AS appears in all paths, extra random connections should be established. A recent modification in the Bitcoin core [27] allows nodes to avoid connecting to multiple nodes in the same AS. While this is a beneficial choice, it does not completely eliminate the routing attack vector

as it only reduces the power of the hosting AS but ignores all other ASes in the AS path.

Use UDP heartbeats and unsolicited information A TCP connection between two Bitcoin nodes may take different forward and backward paths due to asymmetric routing. As a result, each connection traverses more ASes compared to if routes were symmetric, allowing more ASes to observe or modify the exchanged messages. In addition to TCP connections, Bitcoin clients could periodically send UDP messages with unsolicited data *e.g.*, new transactions and blocks. These UDP messages can be used as a heartbeat that will allow nodes to reduce the number of ASes that can access their traffic. Doing so will allow nodes to evade eavesdroppers or discover that a connection to a given peer is intercepted by an attacker (in the opposite direction).

Routing-aware transaction advertisement Obfuscating the first-ever propagation of a transaction each node propagates is key for all attacks against anonymity, including traditional ones [73, 42, 39]. Thus, the Bitcoin Core has adopted diffusion and Dandelion [53] (see § 1.2). While these improvements do not account for AS-level adversaries, we could adapt them accordingly. In diffusion, one could increase the delay for clients whose path contains a frequently-intercepted AS or IXP. In the Dandelion protocol, the selection of the nodes in the stem phase could (in addition to the current criteria) take the AS-path into account such that the created traffic does not often traverse the same AS or IXP.

7.2 Deployment strategies

Cross-layer monitoring Nodes should monitor network- and application-layer statistics to detect abrupt changes. Concerning network-layer statistics, nodes could monitor propagation delay, loss rates, and throughput per connection. Abrupt changes in those metrics indicate the presence of a network attacker which analyzes, stores, deprioritizes or modifies traffic, especially when these changes do not manfest in other applications.

Concerning application-layer statistics, nodes should monitor the distribution of their connections, the time elapsed between a request and the corresponding answer, and the time difference between peer disconnections. A node could respond to such changes by spurring extra random connections or seizing

operations. Thus, the method used to detect changes is crucial for the practical success of this countermeasure.

Using Tor or VPN services Oftentimes passive attacks are targeted to or dependent upon the victim's IP. For instance, the goal of the perimeter attacker is to link transactions to IP addresses. Thus, if a client manages to obfuscate its IP address by using Tor or a VPN service, it should be protected against some of the potential attacks. Still, these approaches are not a panacea. On the one hand, Tor is anonymous by design but has performance and security limitations. More specifically, a network adversary can easily prevent the client from using Tor either by exploiting the DoS mechanism [41] or by merely dropping the corresponding traffic. Observe that the latter is possible as the IPs of all Tor relays are publicly known, and the adversary might intercept the corresponding connections. Even if the client manages to use Tor, it would still be vulnerable to deanonymization by a network attacker that leverages timing analysis [94]. On the other hand, using a VPN service would be an effective countermeasure if the VPN provider is trustworthy. Still, an attacker would be able to map the victim's transactions to the VPN provider's IP. Thus, it is critical that the attacker cannot also trivially map the VPN provider's IP and the victim's identity.

Routing-aware deployment Not all locations in the Internet are equally vulnerable to routing attacks. For instance, hosting a node in an AS that hosts multiple other Bictoin nodes is more secure from the routing perspective. That is the case as connections within the AS are only vulnerable to the hosting AS itself. Similarly, hosting a node in a well-connected AS offers higher path diversity and thus less exposure to AS-level attackers.

Obfuscating the client's state One of the key features used to deanonymize Bitcoin clients in the perimeter attack is whether the victim requested (or received) a particular transaction from any of its observed peers. By requesting a transaction, the client reveals to a networking attacker (or potentially malicious client) that they do not know about a transaction and thus that they have not created it. As a result, the adversary can safely exclude some transactions, effectively decreasing the initial anonymity set. This is also true for the Ethereum geth client [9]. To avoid this, a client should also request the transactions it creates from peers that advertise them. While by doing so, the client increases its load without learning anything new, it also deprives potential attackers of a highly effective feature. Notably, obfuscating the transactions a client knows by requesting them shares a similar method with obfuscating the transactions a light client is interested in by requesting more transactions (i.e., using Bloom Filters in Bitcoin's BIP37 [64]). Instead of requesting more transactions to obfuscate its state, a client can achieve a similar effect by carefully selecting

the peer from which it requests an unknown transaction. Particularly, a client should request transactions in a routing-aware manner, meaning avoid requesting multiple transactions from clients whose connections are intercepted by the same AS or IXP. As a result, an adversary is unlikely to have an accurate view of which transactions the victim knew.

Using fake peers perimeter generalizes to encrypted cryptocurrencies (e.g., Ethereum) because of the networking footprint of its clients. Particularly, a perimeter attacker can infer a client's peers by eavesdropping on this client's connections. Inferring a client's peers is critical for its deanonymization [73]. To shield against this attack, a client should establish connections to "fake" peers with which it does not interact in practice, effectively deceiving a potential attacker into connecting to irrelevant clients. To that end, the client should not request or store any transaction from fake peers; neither should it advertise new transactions to them. As a result, the client obfuscates its footprint from a networking attacker.

8

Conclusions and open problems

In this thesis, we shed light on the impact of routing attacks in cryptocurrencies and Bitcoin in particular. To that end, we first perform a comprehensive routing analysis on the Bitcoin network. We reveal that Bitcoin is heavily centralized from the routing perspective. This finding contradicts one of the most fundamental assumptions made in Bitcoin, namely that it is decentralized, effectively challenging all its security properties. Next, we reveal three concrete and novel routing attacks and evaluate their practicality and effectiveness in the wild. We find that a routing attacker can *(i)* partition the Bitcoin network into two components, effectively breaking consensus; *(ii)* delay information propagation, effectively breaking availability; and *(iii)* deanonymize users, effectively breaking pseudonymity. While these attacks differ with respect to their attacker's goal, they all demonstrate the security risks that a routing attacker imposes on cryptocurrencies such as Bitcoin. Thus, this thesis stresses the importance of cross-layer awareness and the need to deploy both application-layer and network-layer mitigations. As a first step towards this direction, we present practical countermeasures against routing attacks. First, we describe SABRE, a relay network that can secure cryptocurrencies from the partition attack. Next, we also present few cross-layer countermeasures that could mitigate passive attacks, such as the delay and the perimeter attack.

8.1 Future work

While the focus of this thesis is on cryptocurrencies and particularly Bitcoin, routing attacks affect the security of many other distributed systems working atop the Internet. Indeed we observe similar vulnerabilities in anonymity systems such as Tor [94], certificate authorities [38], and DNS [24]. The root cause of these vulnerabilities is the way we design distributed systems today. Oftentimes, researchers focus on the security of individual layers of the system in isolation. An attacker is free to do exactly the opposite: exploit the interactions between layers. In neglecting the insecurity of the routing infrastructure, application developers underestimate the security risks for their users [93]. In focusing on availability and confidentiality threats, network operators underestimate the security risks to Internet applications. This gap leaves the security and privacy of billions of users at risk. To bridge this gap, we need to revisit our abstractions of both the overlay i.e., the distributed system and the underlay i.e., the Internet infrastructure. Better abstractions would allow us to design and study systems in a cross-layer manner.

Revisiting distributed system abstractions

Our conceptual model for distributed systems is often a graph composed of the system's direct participants running a particular protocol. The simplicity of this abstraction allows researchers to prove multiple security properties for the distributed system under consideration. For instance, researchers study the properties of the system assuming a number of byzantine participants. Still, the abstraction is limited to capturing only the application perspective, effectively limiting us to only consider and study: *(i)* application-layer adversaries; *(ii)* incentives of entities formally participating in the protocol; and *(iii)* independent failures. Worse yet, this abstraction is deceiving as it prevents us from seeing the harsh operational reality, leaving critical practical constraints behind. One straightforward approach to bridge this gap would be to augment the initial model with the IP routers in the Internet paths connecting direct protocol participants. Similarly to protocol participants, routers contribute to the protocol's functionality and they access protocol (meta)information. Such an augmented model would allow us to study the effects of a network adversary. Still, including all IP routers might make the model intractable in state. A more scalable approach to bridge the gap between our conceptual abstraction and the operational reality is to add the AS and IXPs in the communication path of each pair of direct participants. Such a model will allow us to study network attacks

such as correlated failures and increased delay, loss, or bandwidth without loss of generality as all routers of an AS are under the same administrative umbrella.

Revisiting our network-layer abstraction

While we need a new model for distributed systems that accounts for network attackers, we should not always look at the network as a threat. It can also serve as an ally. Indeed, there are so many network properties that we could leverage to secure distributed systems. To unlock this potential, though, we would need to redesign the network abstraction, i.e., recognize beneficial network properties and expose them to the higher protocol layers.

As an intuition, we can use known practices and advancements in *(i)* Internet routing; *(ii)* current infrastructure; *(iii)* available end-to-end protocols; and *(iv)* emerging hardware. First, leveraging inferred economic agreements between ASes will allow us to predict the impact of routing manipulation (BGP hijacking). Using this, we can generate deployment instructions for various protocols to preserve their security properties when run atop the insecure Internet. Second, leveraging the Internet's structure is vital to offer readily deployable solutions. Internet Exchange Point, for instance, can speed up secure deployments as they intercept a considerable fraction of Internet traffic, and they are non-profit thus more agile. Third, leveraging advancements in end-to-end protocols can foster innovation in distributed systems design. For instance, QUIC is a recently-introduced application-layer protocol that allows distributed systems to tailor their communication to their performance and security needs without having root access to the end hosts. Finally, leveraging new networking hardware is key to the practicality of hardware/software co-design. For instance, programmable data planes offer a massive opportunity for us to deploy sophisticated network defenses tailored for a specific application without the need to buy expensive hardware.

A PlanetLab for Blockchain

While revisiting our conceptual model is useful, providing formal cross-layer security guarantees might be impractical. Indeed, modeling the behavior of each individual AS in the Internet is extremely challenging, especially for emerging technologies, such as blockchain applications, whose interaction with the underlying network is not well understood. Still, these interactions must be extensively tested before applications are widely deployed. We need a way to perform Internet-wide experiments: a testbed composed of *both computing and networking devices* distributed worldwide, with allocated IP addresses for testing. Such a testbed will allow cross-layer experiments, ethical competitions between attacker and defender teams while demonstrating operation reality/constraints.

Bibliography

[1] About: What is RIPE Atlas? https://atlas.ripe.net/landing/about/.

[2] Announcing Daily RIPE Atlas data archives. https://labs.ripe.net/Members/petros_gigis/announcing-daily-ripe-atlas-data-archives.

[3] BGP in 2019 – The BGP Table. https://blog.apnic.net/2020/01/14/bgp-in-2019-the-bgp-table/.

[4] Bitcoin Blockchain Statistics. https://blockchain.info/.

[5] Bitcoin Core diffusion delay. https://github.com/bitcoin/bitcoin/blob/da4cbb7927497ca3261c1504c3b85dd3f5800673/src/net_processing.cpp#L3813.

[6] bitnodes. https://bitnodes.21.co/.

[7] FALCON. http://www.falcon-net.org/.

[8] FIBRE. http://bitcoinfibre.org/.

[9] Go Ethereum: Official Go implementation of the Ethereum protocol. https://github.com/ethereum/go-ethereum.

[10] Google Internet Traffic Wasn't Hijacked, But It Was Out of Control . https://www.wired.com/story/google-internet-traffic-china-russia-rerouted/.

[11] GoPacket. https://github.com/google/gopacket.

[12] RFC 1267 – Border Gateway Protocol 3 (BGP-3). https://tools.ietf.org/html/rfc1267.

[13] RIPE RIS Raw Data. https://www.ripe.net/data-tools/stats/ris/ris-raw-data.

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

[15] Scapy. http://www.secdev.org/projects/scapy/.

[16] The Bitcoin Relay Network. http://bitcoinrelaynetwork.org/.

[17] The Relay Network. http://bitcoinrelaynetwork.org/.

[18] The RLPx Transport Protocol. https://github.com/ethereum/devp2p/blob/master/rlpx.md.

[19] Transaction surveillance company. https://en.bitcoin.it/wiki/Transaction_surveillance_company.

[20] Amazon EC2 F1 Instances are now available in AWS GovCloud, 2017. https://aws.amazon.com/about-aws/whats-new/2017/11/amazon-ec2-f1-instances-are-now-available-in-aws-govcloud--us/.

[21] Barefoot Tofino Switches: The Technology, 2018. https://www.barefootnetworks.com/technology/.

[22] Bitnodes Statistics, 2018. https://bitnodes.earn.com/.

[23] EC2Instances.info Easy Amazon EC2 Instance Comparison., 2018. https://www.ec2instances.info/?selected=f1.2xlarge,f1.4xlarge.

[24] Hackers emptied Ethereum wallets by breaking the basic infrastructure of the internet, 2018. https://www.theverge.com/2018/4/24/17275982/myetherwallet-hack-bgp-dns-hijacking-stolen-ethereum.

[25] How Bitcoin Mining/Block rewards work., 2018. https://www.anythingcrypto.com/guides/bitcoin-mining-block-rewards-2018.

[26] P4 Behavioral Model, 2018. .https://github.com/p4lang.

[27] Bitcoin Core PR Review Club, 2019. https://bitcoincore.reviews/16702.

[28] Ethereum Mainnet Statistics. https://www.ethernodes.org, 2020.

[29] GLOBAL BITCOIN NODES DISTRIBUTION. https://bitnodes.io/, 2020.

[30] Number of Blockchain wallet users worldwide. https://www.statista.com/statistics/647374/worldwide-blockchain-wallet-users/, 2020.

[31] Propagation of Transactions and Blocks. https://dsn.tm.kit.edu/bitcoin/#propagation, 2020.

[32] Today's Cryptocurrency Prices by Market Cap . https://coinmarketcap.com/, 2020.

[33] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a large european ixp. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 163–174. ACM, 2012. doi:10.1145/2342356.2342393.

[34] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[35] Andreas M. Antonopoulos. The bitcoin network. In *Mastering Bitcoin*, chapter 6. O'Reilly Media, Inc., 2013.

[36] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017.

[37] Rob Austein, Steven Bellovin, Randy Bush, Russ Housley, Matt Lepinski, Stephen Kent, Warren Kumari, Doug Montgomery, Kotikalapudi Sriram, and Samuel Weiler. BGPSEC protocol.

[38] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. Bamboozling certificate authorities with {BGP}. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 833–849, 2018.

[39] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *CCS '14*.

[40] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 15–29, 2014. URL: http://doi.acm.org/10.1145/2660267.2660379, doi:https://doi.org/10.1145/2660267.2660379.

[41] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 122–134. IEEE, 2015.

[42] Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *EuroS&P '19*.

[43] Alexandra Boldyreva and Robert Lychev. Provable Security of S-BGP and Other Path Vector Protocols: Model, Analysis and Extensions. CCS '12, pages 541–552, New York, NY, USA, 2012. ACM. doi:10.1145/2382196.2382254.

[44] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[45] Randy Bush and Rob Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810, January 2013. URL: https://rfc-editor.org/rfc/rfc6810.txt, doi:10.17487/RFC6810.

[46] The CAIDA AS relationship dataset - 20191001. http://data.caida.org/datasets/as-relationships/serial-1/.

[47] The caida ixps dataset - 201910. http://data.caida.org/datasets/ixps/ix-asns_201910.jsonl. Accessed: 2020-03-12.

[48] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Xuemin Lin, Chengfei Liu, and Weifa Liang. Efficiently computing k-edge connected components via graph decomposition. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 205–216, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2463676.2465323, doi:10.1145/2463676.2465323.

[49] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[50] Christian Decker and Roger Wattenhofer. *Bitcoin Transaction Malleability and MtGox*, pages 313–326. Springer International Publishing, Cham, 2014. URL: http://dx.doi.org/10.1007/978-3-319-11212-1_18, doi:10.1007/978-3-319-11212-1_18.

[51] Ittay Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE, 2015.

[52] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[53] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *POMACS*, 2018.

[54] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998. URL: http://doi.acm.org/10.1145/285055.285059, doi:10.1145/285055.285059.

[55] Lixin Gao and Jennifer Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking (TON)*, 9(6):681–692, 2001.

[56] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[57] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 692–705, New York, NY, USA, 2015. ACM. doi:10.1145/2810103.2813655.

[58] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. Are we there yet? on rpki's deployment and security. In *NDSS*, 2017.

[59] Phillipa Gill, Michael Schapira, and Sharon Goldberg. Let the Market Drive Deployment: A Strategy for Transitioning to BGP Security. SIGCOMM '11, pages 14–25, New York, NY, USA, 2011. ACM. doi:10.1145/2018436.2018439.

[60] Sharon Goldberg. Why is it taking so long to secure internet routing? *Communications of the ACM*, 57(10):56–63, 2014.

[61] Sharon Goldberg, Michael Schapira, Peter Hummon, and Jennifer Rexford. How secure are secure interdomain routing protocols. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 87–98. ACM, 2010.

[62] Arpit Gupta, Robert MacDavid, Rüdiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, and Laurent Vanbever. An Industrial-Scale Software Defined Internet Exchange Point. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX, 2016.

[63] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean Donovan, Russ Clark, Nick Feamster, Jennifer Rexford, and Scott Shenker. SDX: A Software Defined Internet Exchange. In *Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM'14)*. ACM, 2014.

[64] M Hearn and M Corallo. Connection bloom filtering. bitcoin improvement proposal 37 (2012).

[65] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[66] Yih-Chun Hu, Adrian Perrig, and Marvin Sirbu. SPV: Secure Path Vector Routing for Securing BGP. SIGCOMM '04, pages 179–192, New York, NY, USA, 2004. ACM. doi:10.1145/1015467.1015488.

[67] Theo Jepsen, Daniel Alvarez, Nate Foster, Changhoon Kim, Jeongkeun Lee, Masoud Moshref, and Robert Soulé. Fast string searching on pisa. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 21–28, 2019.

[68] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. Netchain: Scale-free sub-rtt coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/jin.

[69] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136. ACM, 2017.

[70] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[71] Ghassan O Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):1–32, 2015.

[72] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. Pretty Good BGP: Improving BGP by Cautiously Adopting Routes. In *Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, ICNP '06, pages 290–299, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/ICNP.2006.320179.

[73] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer, 2014.

[74] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. Citeseer.

[75] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[76] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[77] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[78] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and KC Claffy. As relationships, customer cones, and validation. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 243–256, 2013.

[79] Robert Lychev, Sharon Goldberg, and Michael Schapira. BGP Security in Partial Deployment: Is the Juice Worth the Squeeze? In *SIGCOMM*, 2013.

[80] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140, 2013.

[81] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes.

[82] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. *IACR Cryptology ePrint Archive*, 2015:796, 2015.

[83] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *IFIP/IEEE International Symposium on Internet Management*, pages 1327–1332. IEEE, 2015.

[84] P.C. van Oorschot, Tao Wan, and Evangelos Kranakis. On interdomain routing security and pretty secure bgp (psbgp). *ACM Trans. Inf. Syst. Secur.*, 10(3), July 2007. doi:10.1145/1266977.1266980.

[85] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks, 2016.

[86] Alex Pilosov and Tony Kapela. Stealing The Internet. An Internet-Scale Man In The Middle Attack. DEFCON 16.

[87] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. URL: http://www.ietf.org/rfc/rfc4271.txt.

[88] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[89] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.

[90] Brandon Schlinker, Kyriakos Zarifis, Italo Cunha, Nick Feamster, and Ethan Katz-Bassett. Peering: An as for us. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 18:1–18:7, New York, NY, USA, 2014. ACM. doi:10.1145/2670518.2673887.

[91] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176. ACM, 2017.

[92] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[93] Yixin Sun, Maria Apostolaki, Henry Birge-Lee, Laurent Vanbever, Jennifer Rexford, Mung Chiang, and Prateek Mittal. Securing internet applications from routing attacks, 2020. arXiv:2004.09063.

[94] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. Raptor: Routing attacks on privacy in tor. In *USENIX Security Symposium*, pages 271–286, 2015.

[95] Andree Tonk. Large scale BGP hijack out of India, 2015. http://www.bgpmon.net/large-scale-bgp-hijack-out-of-india/.

[96] Andree Tonk. Massive route leak causes Internet slowdown, 2015. http://www.bgpmon.net/massive-route-leak-cause-internet-slowdown/.

[97] Russ White. Securing bgp through secure origin bgp (sobgp). *Business Communications Review*, 33(5):47–47, 2003.