Erlay: Efficient Transaction Relay in Bitcoin

by

Hlib Naumenko

Bachelor in Software Engineering, Kharkiv National University of Radio Electronics, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

April 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Erlay: Efficient Transaction Relay in Bitcoin

submitted by **Hlib Naumenko** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Ivan Beschastnikh, Computer Science
Co-supervisor

Alexandra Fedorova, Electrical and Computer Engineering
Co-supervisor

Norm Hutchinson, Computer Science
Supervisory Committee Member

Abstract

Bitcoin is a top-ranked cryptocurrency that has experienced huge growth and survived numerous attacks. The protocols making up Bitcoin must therefore accommodate the growth of the network and ensure security. However, Bitcoin's transaction dissemination protocol has mostly evaded optimization. This protocol is based on flooding and though it is secure and fault-tolerant, it is also highly inefficient. Specifically, our measurements indicate that 43% of the traffic generated by transaction dissemination in the Bitcoin network is redundant.

In this thesis we introduce a new transaction dissemination protocol called *Erlay*. Erlay is a hybrid protocol that combines limited flooding with intermittent reconciliation. We evaluated Erlay in simulation and by implementing and deploying it at scale. Compared to Bitcoin's current protocols, Erlay reduces the bandwidth used to announce transactions by 84% without significantly affecting privacy or propagation speed. In addition, Erlay retains the existing Bitcoin security guarantees and is more scalable relative to the number of nodes in the network and their connectivity. Erlay is currently being investigated by the Bitcoin community for future use with the Bitcoin protocol.

Lay Summary

Bitcoin is a peer-to-peer electronic cash system, which operates over computers (nodes) running Bitcoin software across the world. Those nodes exchange messages, which represent transactions (transfers of funds) as well as additional information.

In currently deployed software, Bitcoin transactions are relayed across all nodes via *flooding*, where every node notifies all its peers of a new transaction. While this approach is robust and has fast transaction relay, it is inefficient in terms of consumed bandwidth, because every node learns about every transaction multiple times.

We design Erlay, a transaction relay protocol, which is more efficient than flooding. Erlay is a combination of rapid low-fanout flooding and efficient set reconciliation. We design and evaluate Erlay, and show that it outperforms the current protocol and may enable better security and privacy of Bitcoin transactions in the future.

Preface

All of the work presented henceforth was conducted in the NSS (Networks, Systems and Security) lab in the Department of Computer Science at the University of British Columbia, Point Grey campus.

The work presented in this thesis is original, unpublished work by the author, Hlib Naumenko. This work was performed in collaboration with Gregory Maxwell and Pieter Wuille. The entirety of this work was designed and written in assistance with Dr. Ivan Beschastnikh and Dr. Alexandra Fedorova, who supervised this projects.

The design of the Erlay protocol, simulator and reference implementation were performed by the author. Gregory Maxwell is the author of the idea of mempool reconciliation, which pre-dated Erlay. Pieter Wuille is the main contributor (along with Gregory Maxwell and the author) of MiniSketch [1], an underlying library developed for Erlay. Both Gregory Maxwell and Pieter Wuille contributed to the discussions around implementation details of Erlay.

[1] https://github.com/sipa/minisketch

Table of Contents

List of Tables

List of Figures

Glossary

BTC Bitcoin

DHT distributed hash table

DOS denial of service

LOC lines of code

P2P peer-to-peer

Acknowledgments

This research was funded by a grant from the Natural Science and Engineering Research Council of Canada (NSERC) Discovery Grant.

Firstly, I would like to thank Ivan Beschastnikh for supervising me along this journey, providing insights on conducting research and allowing me to freely choose my research direction. I would like to thank Alexandra Fedorova for joining our work as a co-supervisor and contributing while going through a critical personal mission in her life.

I would also like to thank Blockstream for providing me with an opportunity of an internship, where I was able to expand my research, and, even more importantly, to obtain experience in Bitcoin Core contributions. I am also grateful to them for supporting my work after the end of my internship and expensing my trips to conferences.

I would also like to thank the Bitcoin community for warm welcoming, interest in my work and useful feedback.

Chapter 1

Introduction

Bitcoin is a peer-to-peer (P2P) trustless electronic cash system [47]. Recent estimates indicate that there are just over 60,000 nodes in the Bitcoin network[1]. And, to keep up with the growth in the number of nodes and usage of the network, the system must be continually optimized while retaining the security guarantees that its users have come to expect. For example, prior work has considered optimizations to Proof-of-Work [7, 61], moving payments *off-chain* [53], improving transaction throughput by increasing the block size [5, 26, 60], as well as optimizing block relay [13, 34, 51] and transaction representation within a block [37]. One reason why optimizing Bitcoin is challenging is because security is not just a theoretical concern: a wide variety of attacks have been published [6, 8, 9, 14, 17, 18, 20, 27, 30, 33, 35, 39, 42–44, 48, 50] and the Bitcoin software is being updated to protect against these. Therefore, any modification to the system must carefully consider the security implications of the change.

One bottleneck in the Bitcoin network that has not received much attention is the cost of transaction relay. A Bitcoin *transaction* corresponds to a transfer of funds between several accounts. Fig. 1.1 overviews the lifecycle of a transaction in the Bitcoin network. To be accepted by the network of nodes, a transaction must be first disseminated, or relayed, throughout the network. Then it must be validated and included into a *block* with other valid transactions. Finally, the block containing the transaction must be relayed to all the nodes. Every Bitcoin transaction must

[1] https://luke.dashjr.org/programs/bitcoin/files/charts/software.html

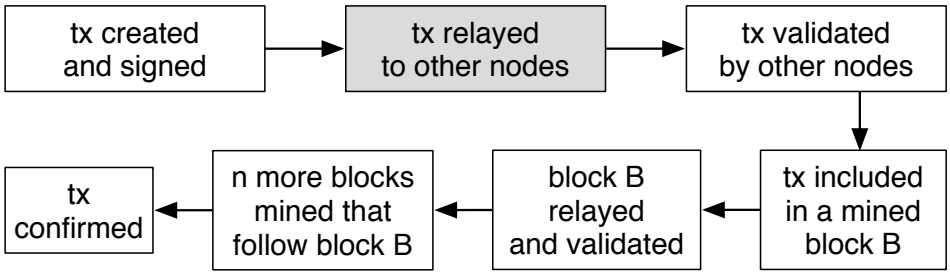

Figure 1.1: Lifecycle of a Bitcoin transaction. In this thesis we optimize the protocols for relaying transactions between nodes in the Bitcoin network (grey box).

reach almost all nodes in the network, and prior work has demonstrated that full coverage of the network is important for it to preserve security [57].

Today, Bitcoin relays transactions using a protocol based on *flooding*: every message received by a node is transmitted to all of its neighbors. Flooding has high fault-tolerance since no single point of failure will halt relay, and it has low latency since nodes learn about transactions as fast as possible [38].

However, flooding has poor bandwidth efficiency: every node in the network learns about the transaction multiple times. Our empirical measurements demonstrate that transaction announcements account for 30–50% of the overall Bitcoin traffic. This inefficiency is an important scalability limitation: the inefficiency increases as the network becomes more connected, while connectivity of the network is desirable to the growth and the security of the network.

Prior work has explored two principal approaches to this bandwidth inefficiency. The first is the use of short transaction identifiers (to decrease message size) [32]. The second is to exclusively use blocks and never transmit individual transactions [40]. Both approaches are inadequate: short identifiers only reduce the constant factor and do not scale with the connectivity of the network, while using only blocks will create spikes in block relay and transaction validation. We discuss these approaches further in Section 11.

The contribution of this thesis is Erlay, a new protocol that we designed to optimize Bitcoin's transaction relay while maintaining the existing security guarantees. The main idea behind our protocol is to reduce the amount of information

propagated via flooding and instead use an efficient set reconciliation method [45] for most of the transaction dissemination. In addition we design this protocol to withstand DoS, timing, and other attacks.

We implemented Erlay in a simulator as well as part of the mainline Bitcoin node software and evaluated Erlay at scale. Our results show that Erlay makes announcement-related bandwidth negligent while introducing a 50% latency increase.

In summary, this thesis makes the following contributions:

- We analyze bandwidth inefficiency of Bitcoin's transaction relay protocol. We do this by running a node connected to the Bitcoin network as well as by running a simulation of the Bitcoin network. Our results demonstrate that 88% of the bandwidth used to announce transactions (and around 40% of the overall bandwidth) are redundant.

- We propose a new, bandwidth-efficient, transaction relay protocol for Bitcoin called *Erlay*, which is a combination of fast low-fanout flooding and efficient set reconciliation, designed to work under the assumptions of the Bitcoin network.

- We demonstrate that the protocol achieves a close to optimal combination of resource consumption and propagation delay, and is robust to attacks. Erlay reduces the bandwidth used to relay transactions by 48% immediately, and allows the Bitcoin network to achieve higher connectivity to improve the security of the network.

Chapter 2

Background

There are 2 types of nodes in the Bitcoin network: **private nodes** that do not accept inbound connections and **public nodes** that *do* accept inbound connections (see Fig. 2.1). Public nodes act as a backbone of the network: they help new nodes bootstrap onto the network. If a person does not want to contribute resources (e.g., bandwidth) to the network, then they run a private node, which only connects to the existing public nodes. Once they joined the network, public and private nodes are indistinguishable in their operation: both perform transaction and block validation and relay valid transactions and blocks to their peers.

The current version of the Bitcoin transaction relay protocol propagates messages among nodes using *diffusion* [1], which is a variation on random flooding. Flooding is a protocol where each node announces every transaction it receives to each of its peers. Announcements can be sent on either inbound and outbound links. With diffusion a peer injects a random delay before announcing a received transaction to its peers. This mitigates timing attacks [50] and significantly reduces the probability of in-flight collisions (when 2 nodes simultaneously announce the same transaction over the link between them).

The protocol by which a transaction propagates between two peers is illustrated in Fig. 2.2. When a Bitcoin node receives a transaction (peer 1 in Fig. 2.2), it advertises the transaction to all of its peers except to the node that sent the transaction in the first place. To advertise a transaction, a node sends a hash of the transaction within an *inventory*, or *INV* message. If a node (peer 2 in Fig. 2.2) hears

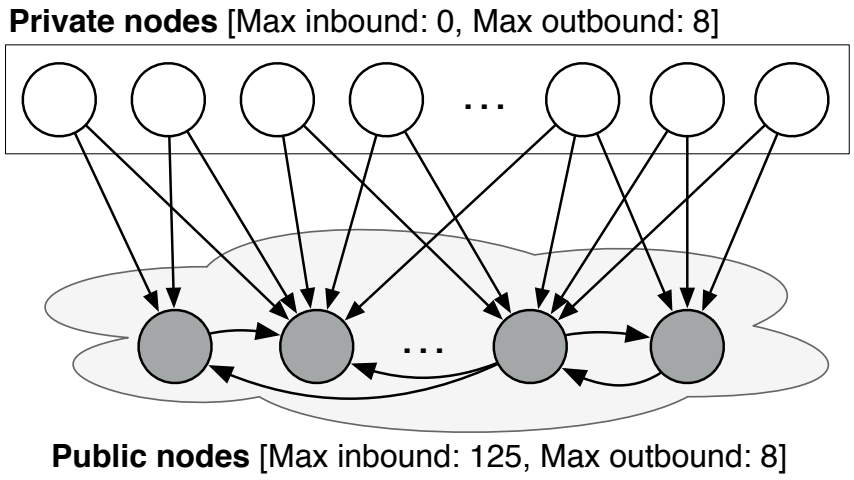

Figure 2.1: Private and public nodes in the Bitcoin network.

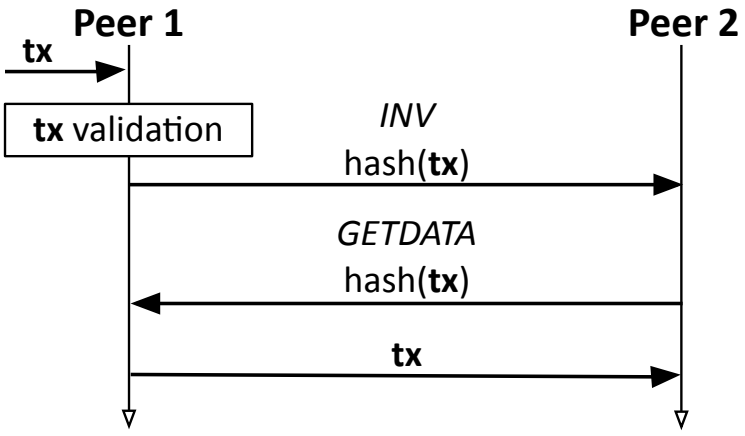

Figure 2.2: Transaction exchange between two peers.

about a transaction for the first time, it will request the full transaction by sending a *GETDATA* message to the node that sent it the *INV* message.

We refer to the transaction-advertising portion of the protocol (all the INV messages) as *BTCFlood*. Since it relies on flooding, most transactions are advertised through each link in the network in one direction (except those that are advertised during the block relay phase). As a result, a node with n connections will send and

receive between n and $2n$ INV messages for a single transaction (two nodes may announce the same transaction simultaneously to each other).

Both public and private nodes limit the number of inbound and outbound connections (Fig. 2.1). By default a private node has no inbound connections and up to 8 outbound connections, while a public node can have 8 outbound connections as well as up to 125 inbound connections (but the limit can be configured up to around 1,000). Thus, as the Bitcoin network grows, the bandwidth and computational requirements to run a public node quickly increase. This is because private nodes connect to multiple public nodes to ensure that they are connected to the network through more than a single peer.

As a result, Bitcoin designers have focused on (1) making the running of a public node more accessible, in terms of required bandwidth, computational power, and hardware resources; and, (2) making public nodes more efficient so that they can accept more connections from private nodes. Our work targets both objectives.

Chapter 3

The problem with flooding transactions

Flooding is inefficient. BTCFlood sends many redundant transaction announcements. To see why, let us first consider how many announcements would be sent if the protocol were efficient. Since, optimally, each node would receive each announcement exactly once, *the number of times each announcement is sent should be equal to the number of nodes*.

Next, let us consider how many times an announcement is sent with BTCFlood. By definition, each node relays an announcement on each of the links except the one where that announceent originally arrived. In other words, each link sees each announcement once, if no two nodes ever send the same announcement to each other simultaneously, and more than once if they do. Therefore, *in BTCFlood each announcement is sent at least as many times as the number of links*.

If N is the number of nodes in the Bitcoin network, the number of links is $8N$, because each node must make eight outbound connections. Therefore, the number of *redundant* announcements is at least $8N - N = 7N$. Each announcement takes 32 bytes out of 300 total bytes needed to relay a single transaction to one node. (These 300 bytes include the announcement, the response and the full transaction body). Therefore, if at least seven out of eight announcements are redundant (corresponding to 224 bytes), at least 43% of all announcement traffic is wasteful.

We validated this analysis experimentally. We configured a public Bitcoin node

with eight outbound connections and ran it for one week. During this time, our node also received four inbound connections. We measured the bandwidth dedicated to transaction announcements and other transaction dissemination traffic. A received announcement was considered redundant if it corresponded to an already known transaction. A sent announcement was considered redundant if it was not followed by a transaction request. According to our measurements 10% of the traffic corresponding to received announcements and 95% of the traffic corresponding to the sent announcements was redundant. Overall, 55% of all traffic used by our node was redundant.

Higher connectivity requires more bandwidth. Given that the amount of redundant traffic is proportional to the number of links, increasing the connectivity of the network (the number of outbound links per node) linearly increases bandwidth consumption in BTCFlood.

We modeled how the bandwidth consumption of disseminating one transaction across the network of 60K nodes increases with connectivity. Fig. 3.1 (whose results we confirmed via simulation) shows that announcement traffic turns dominant as the network becomes more connected.

Higher connectivity offers more security. In peer-to-peer (P2P) networks, higher connectivity improves network security. This was demonstrated by both traditional P2P research [3, 4] and Bitcoin-specific prior work [8, 16, 30, 39, 49].

Certain attacks become less successful if the network is highly connected [29, 39, 50]. As an example, the eclipse attack [30] has shown that fewer than 13 connections would be detrimental to security of the network. As another example, consider a recently discovered vulnerability [18] that relies on *InvBlock* [44]. InvBlock is a technique that prevents a transaction from being propagated by first announcing it to a node, but then withholding the transaction contents for two minutes. With higher connectivity, this attack is easier to mitigate. For that reason, Bitcoin literature repeatedly recommends to increase the number of connections to make the network more robust [8, 16].

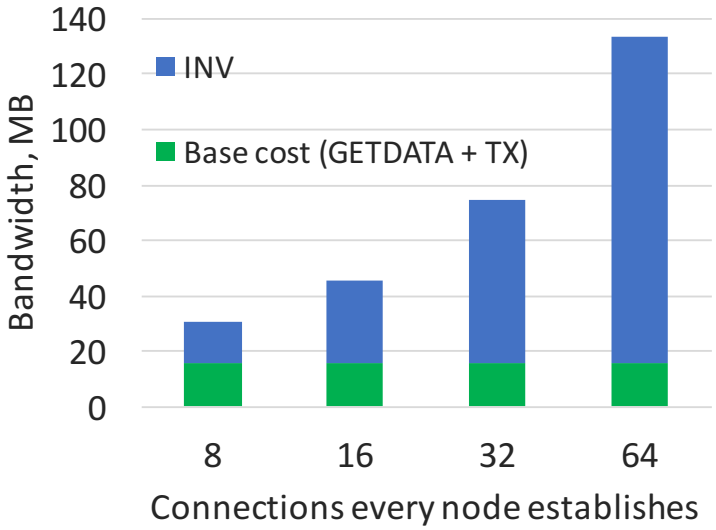

Figure 3.1: Analytical cost of relaying one transaction across the network of 60,000 nodes.

Chapter 4

Protocol requirements

R1: Good scalability as the number of connections increases. Our main goal is to design a transaction dissemination protocol that has good scalability in function of the number of connections. This way, we can make the network more secure without sacrificing performance.

R2: Maintain a network topology suited for a trustless environment. Bitcoin's premise of a trustless environment puts constraints on the design of its network. Although imposing a structure onto a network, e.g., by organizing it into a *tree* or *star* topology, or by using distributed hash table (DHT)-style routing, enables bandwidth-efficient implementation of flooding, this also introduces the risks of censorship or partitioning [39]. The topology of the network must, therefore, remain unstructured, and routing decisions must be made independently by every node based on their local state.

R3: Maintain a reasonble latency. Transaction propagation delays should remain in the ballpark of those experienced with the existing protocol. Low latency is essential to user experience and enables better efficiency in block relay [13].

R4: Be robust to attacks under the existing threat model. Our protocol must remain robust under the same threat model as that assumed by the existing protocol. Similarly to Bitcoin, we assume that an attacker has control over a limited number of nodes in the network, has a limited ability to make other nodes to connect to it, and is otherwise unrestricted in intercepting and generating traffic for peers that it is connected to.

The transaction relay protocol must not be any more susceptible to DoS attacks and client deanonymization, and must not leak any more information about the network topology [50] than the existing protocol.

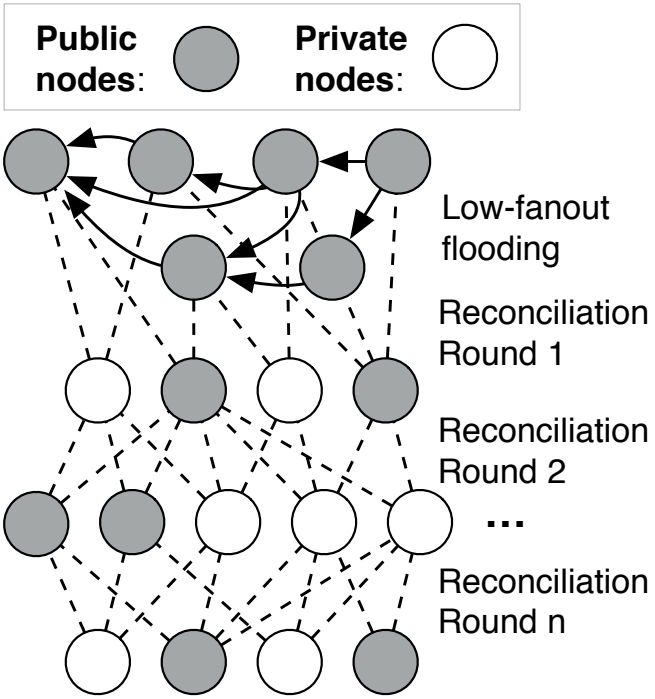

Figure 4.1: Erlay disseminates transactions using low-fanout flooding as the first step, and then several rounds of reconciliation to reach all nodes in the network.

Chapter 5

Erlay design

Traditionally, P2P networks addressed inefficiency of flooding by imposing a structured overlay onto an ad-hoc topology. We refrained from structured network organizations for security reasons discussed in Section 4. Instead, our design relies on two common system-building techniques: delay and batching.

Instead of announcing every transaction on each link, a node using our protocol advertises it to a subset of peers – this is called *low-fanout flooding*. To make sure that all transactions reach the entire network, a node periodically sends to its peers a batch of transaction announcements that it accumulated recently, and the peers request those transactions that are missing from their own collections. This is called *set reconciliation*. Our protocol, that is comprised of low-fanout flooding and set reconciliation (Fig. 4.1), is called Erlay.

Low-fanout flooding. The rationale behind low-fanout flooding is to expediently relay a transaction to be within a small number of hops from every node in the network. If each transaction ends up close to every node, then reconciliation can finish dissemination using a small number of rounds. Therefore, a key decision in low-fanout flooding is to which peers to relay.

Set reconciliation. *Set reconciliation* was proposed as an alternative to synchronization in distributed systems [45]. Using set reconciliation a node in a P2P network periodically compares its local state to the state of its peers, and sends/requests only the necessary information (the state difference). Set reconciliation may be viewed as an efficient version of *batching* (accumulating multiple state updates

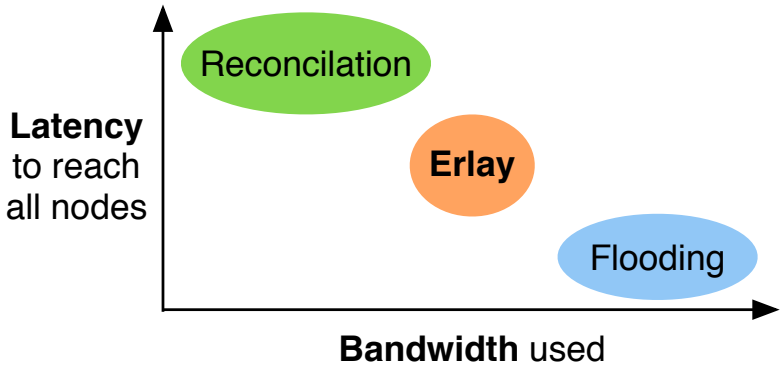

Figure 5.1: Comparison of reconciliation, flooding, and Erlay in their bandwidth usage and latency to reach all nodes.

and sending them as a single message). The key challenge in practical reconciliation is for the peers to efficiently compute their missing transaction state, and to limit the exchanged transactions to just those that the other peer is missing.

Fig. 5.1 shows how Erlay attempts to find a sweet spot in terms of bandwidth and latency by combining flooding, which wastes bandwidth but disseminates transactions quickly, and reconciliation, which takes longer, but does not waste bandwidth.

5.0.1 Low fanout flooding

Flooding is expensive, so we want to use it sparingly and in *strategic* locations. For that reason, only well-connected public nodes flood transactions to other public nodes via outbound connections. Since every private node is directly connected to several public nodes, this policy ensures that a transaction is quickly propagated to be within one hop from the majority of the nodes in the network. As a result, only one or two reconciliation rounds are needed for full reachability (**R3**). According to this, the protocol we propose may be viewed as two-tier optimistic replication [55].

To meet our scalability goal (**R1**), we limit the flooding done by public nodes to eight outbound connections even if the total number of these connections is higher. This way, increasing connectivity does not increase transaction dissemination cost proportionally.

The decision to relay through outbound connections, but not the inbound ones, was made to defend against timing attacks [18, 50]. In a timing attack, an attacker connects to a victim and listens to all transactions that a victim might send on that link (the inbound connection for the victim). If an attacker learns about a transaction from multiple nodes (including the victim), the timing of transaction arrival can be used to guess whether a transaction originated at the victim: if it did then it will most likely arrive from the victim earlier than from other nodes. BTCFlood introduces a diffusion delay to prevent timing attacks. In Erlay, since we do not forward individuial transactions to inbound links, this delay is not necessary. So this decision favors both **R3** and **R4**.

Transactions in the Bitcoin (BTC) network may originate both at public and private nodes. Public nodes disseminate their transactions to a subset of public nodes via flooding, and via reconciliation to private nodes (and the remaining public nodes). In the protocol we propose, private nodes do not relay transactions via flooding, so the network will learn about their originating transactions via reconciliation: private nodes will add their own transactions to the batch of other transactions that they are forwarding to their peer during reconciliation. Since a private node forwards its own transactions as part of the batch, as opposed to individually, a malicious public node is unlikely to discover the origin of a transaction (**R4**).

5.0.2 Set reconciliation

In Erlay peers perform set reconciliation by computing a local *set sketch*, as defined by the PinSketch algorithm [19]. A set sketch is a type of set checksum with two important properties:

- Sketches have a predetermined capacity, and when the number of elements in the set does not exceed the capacity, it is always possible to recover the entire set from the sketch by *decoding* the sketch. A sketch of b-bit elements with capacity c can be stored in bc bits.
- A sketch of the symmetric difference between the two sets (i.e., all elements that occur in one but not both input sets), can be obtained by XORing the bit representation of sketches of those sets.

These properties make sketches appropriate for a bandwidth-efficient set reconciliation protocol. More specifically, if Alice and Bob suspect that their sets largely, but not entirely overlap, they can use the following protocol to have both parties learn all the elements of the two sets:

- Alice, Bob both locally compute sketches of their sets.
- Alice sends her sketch to Bob.
- Bob combines the two sketches, and obtains a sketch of the symmetric difference.
- Bob tries to recover the elements from the symmetric difference sketch.
- Bob sends to Alice the elements that she is missing.

This procedure will always succeed when the size of the difference (elements that Alice has but Bob does not have plus elements that Bob has but Alice does not have) does not exceed the capacity of the sketch that Alice sent. A key property of this process is that it works regardless of the actual set sizes: only the size of the set differences matters.

Decoding the sketch is computationally expensive and is quadratic in the size of the difference. That is why, accurately estimating the size of the difference (Section 5.0.2) and reconciling before the set difference becomes too large (Section 5.0.2) are important goals for the protocol.

Reconciliation round

Fig. 5.2 summarizes the reconciliation protocol. To execute a round of reconciliation, every node maintains a *reconciliation set* for each one of its peers. A reconciliation set consist of short IDs of transactions that a node would have sent to a corresponding peer in regular BTCFlood, but has not because Erlay limits flooding. We will refer to Alice's reconciliation set for Bob as A and Bob's set for Alice as B. Alice and Bob will compute the sketches for these reconciliation sets as described in the previous section.

Important parameters of the protocol are: D – the true size of the set difference, d – an estimate of D, and q – a parameter used to compute d. We provide the derivation of these values below. First, we describe a reconciliation round:

1. According to a chosen reconciliation schedule (Section 5.0.2), Alice sends to Bob the size of A and q.
2. Bob computes d, an estimate of D, between his B and Alice's A (see below).
3. Bob computes a sketch of B which allows to reconcile D transactions and sends it to Alice, along with the size of B.
4. Alice receives Bob's sketch of B, computes a sketch of A, and XORs the two sketches. Now Alice has a sketch of the difference between A and B.
5. If the difference size was estimated correctly, Alice is able to decode the sketch computed in the previous step, request the transactions that she is missing from Bob, and then advertise to Bob the transactions that he is missing. If the estimation was incorrect (sketch decoding failed), Alice will resort to bisection (Section 5.0.2).
6. After this process, Alice updates q (see below) and clears A. Bob clears B.

Accurate estimation of D is crucial for success of reconciliation. Prior work estimated D using techniques like min-wise hashing [11] or random projections [24]. These techniques are complex, and we were concerned that they could end up using more bandwidth that they save. Therefore, we resorted to a minimalistic approach, where we estimate the size of the set difference based on just the current sizes of sets and the difference observed in the previous reconciliation round:

$$d = abs(|A| - |B|) + q \cdot min(|A|, |B|) + c,$$

where q is a floating point coefficient (derived below) that characterizes previous reconciliation, and c is a coefficient for handling special cases.

Indeed, the difference between two sets cannot be smaller than the difference in their sizes. To avoid costly underestimations, we add the size of the smaller

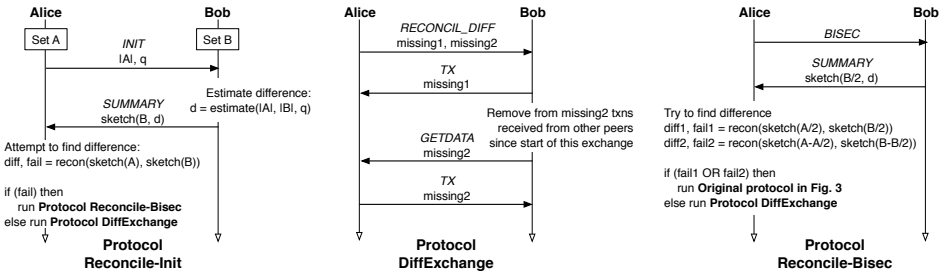

Figure 5.2: Reconciliation protocol with correct difference estimation (Reconcile-Init, followed by DiffExchange). And, reconciliation protocol with incorrect difference estimation (Reconcile-Init, followed by Reconcile-Bisec). In case reconciliation fails during Reconcile-Bisec, reconciliation falls back to Bitcoin's current exchange method (see Fig. 2.2).

set normalized by q, and a constant $c = 1$, which prevents estimating $d = 0$ when $|A| = |B|$ and $q \cdot min(|A|, |B|) = 0$.

The coefficient q characterizes earlier reconciliation, so before the very first reconciliation round it set to zero. At the end of a reconciliation round, we simply update q based on the true D that we discovered during the round, by substituting D for d in the above equation, dropping c and solving for q:

$$q = \frac{D - abs(|A| - |B|)}{min(|A|, |B|)}$$

This updated q will be used in the next reconciliation round. We compute q in this way, because we assume that every node in the network will have a consistent optimal q.

Reconciliation is a fertile ground for DoS attacks, because decoding a sketch is computationally expensive. To prevent these attacks, in our protocol the node that is interested in reconciliation (and the one that has to decode the sketch) initiates reconciliation (Alice, in our example). Bob cannot entice Alice to perform excessive sketch decoding.

Reconciliation schedule

Every node initiates reconciliation with one outbound peer every T seconds. Choosing the right value for T is important for performance and bandwidth consumption. If T is too low, reconciliation will run too often and will use more bandwidth than it saves. If T is too high, reconciliation sets will be large and decoding set differences will be expensive (the computation is quadratic in the number of differences). A large T also increases the latency of transaction propagation.

A node reconciles with one peer every T seconds. Since every node has c outbound connections, every link in the network would, on average, run reconciliation every $T \cdot c$ seconds. This means that the average reconciliation set prior to reconciliation would contain $T \cdot c \cdot TX_{rate}$ transactions, where TX_{rate} is the global transaction rate. This also means that during the interval between reconciliations every node would receive $T \cdot TX_{rate}$ transactions.

We use a value of 1 second for T in Erlay. With this setting, and the current ratio of private to public nodes, every public node will perform about eight reconciliations per second. Given the current maximum Bitcoin network transaction rate TX_{rate} of 7 transactions/s, the average difference set size for this protocol is 7 elements. We evaluate our choice of parameters in Section 8.

Bisection for set difference estimation failure

Our set reconciliation approach relies on the assumption that an upper bound for the set difference between two peers is predictable. That is, if the actual difference is higher than estimated, then reconciliation will fail. This failure is detectable by a client computing the difference. An obvious solution to this failure is to recompute and retransmit the sketch assuming a larger difference in the sets. However, this would make prior reconciliation transmissions useless, which is inefficient.

Instead, Erlay uses reconciliation *bisection*, which re-uses previously transmitted information. Bisection is based on the assumption that elements are uniformly distributed in reconciliation sets (this may be achieved by hashing). If a node is unable to reconstruct set difference from a product of two sketches, the node then makes an additional reconciliation request, similar to the initial one, but this request is applied to only a fraction of possible messages (e.g., to transactions in the

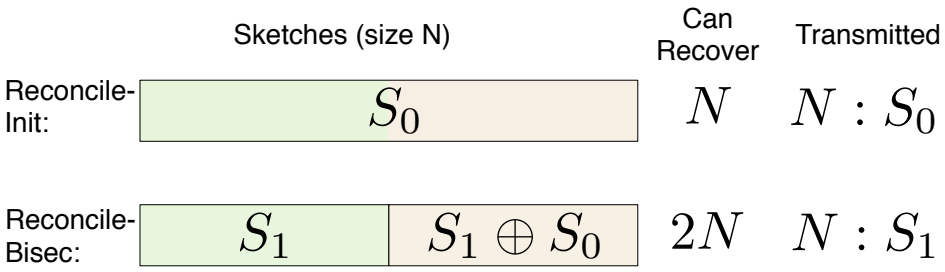

Figure 5.3: Bisection is enabled by the linearity of sketches

range $0x0$–$0x8$). Because of the linearity of sketches, a sketch of a subset of transactions would allow the node to compute a sketch for the remainder, which saves bandwidth.

However, this approach would allow to recover at most $2d$ differences, where d is the estimated set difference in the initial step. Even though bisections are not limited to one and may be applied consequentually without loosing efficiency, in our implementation, after a reconciliation step failure, we allow only one bisection with a new overall estimate $2d$ (see Fig. 5.3). The bisection process is illustrated in protocol Reconcile-Bisec in Figure 5.2.

If bisection fails, then Erlay falls backs to the original INV-GETDATA protocol (Fig. 2.2) and applies it to all of the transactions in two sets being reconciled.

Chapter 6

Implementation details

In this section we describe low-level design decisions required to implement Erlay and increase its bandwidth efficiency (**R2**) and make it robust to *collision-based* denial of service (DOS) attacks (**R4**).

Library implementation. We created a C++ library with 3055 lines of code (LOC)[1], that is an optimized implementation of the PinSketch [19] algorithm. We benchmarked the library to verify that set reconciliation would not create high computational workload on Bitcoin nodes. Fig. 6.1 shows the decoding performance on an Intel Core i7-7820HQ CPU of our library (PinSketch) as compared to CPISync [58][2] for varying difference sizes[3]. Our library has sub-millisecond performance for difference sizes of 100 elements or fewer. As we will show later (Fig. 8.3) this performance is sufficiently fast for the differences we observe in practice (in simulation and in deployment).

We used this library to build a reference implementation of Erlay as a part of the Bitcoin Core software, which we evaluate in Section 9.

Short identifiers and salting. The size of a transaction ID in the Bitcoin protocol is 32 bytes. To use PinSketch [19], we have to use shorter, 64 bit, identifiers. Using fewer bits reduces the bandwidth usage by 75% (**R2**), but it also creates a probability of collisions. Collisions in transaction relay is an attack surface, because a

[1] https://github.com/sipa/minisketch

[2] https://github.com/trachten/cpisync

[3] We omit other library benchmarks due to limited space.

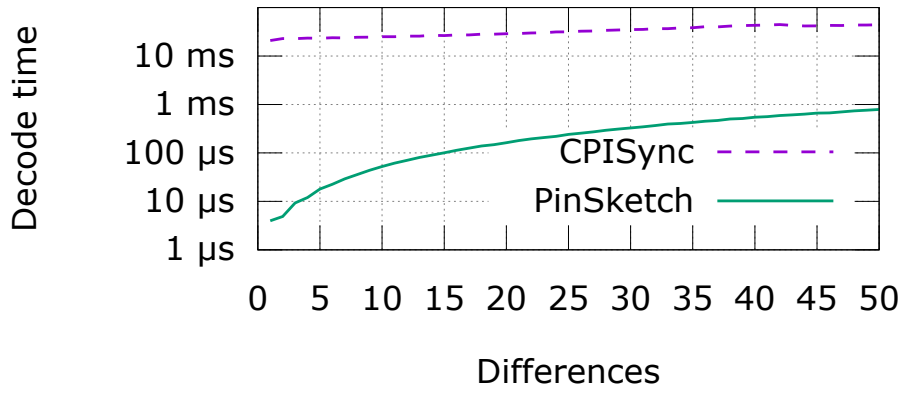

Figure 6.1: The decode time of our library (PinSketch) as compared to CPISync for varying set difference sizes.

malicious actor may flood a network with colluding transactions and fill *memory pools* of the nodes with transactions which would be propagated and confirmed in a very slow manner. Thus, we want to secure the protocol against such attacks (**R4**).

While collisions on one side of communication are easy to detect and handle, collisions involving transactions on both sides may cause a significant slowdown. To mitigate this, we use different salt (random data added to an input of a hash-function) while hashing transaction IDs into short identifiers.

The salt value is enforced by the peer that initiates the connection, and per Erlay's design, requests reconciliation. Since the peer requesting reconciliation also computes the reconciliation difference, the requestor peer would have to deal with short IDs of unknown transactions. Since salt is chosen by the requestor, re-using the same salt for different reconciliations would allow him to compare salted short IDs of unknown transactions to the IDs received during flooding from other peers at the same time.

Low-fanout diffusion delay. Bitcoin flooding mitigates timing attacks [50] and in-flight collisions by introducing a random delay into transaction announcements. For timing attacks Bitcoin assumes that an attacker connects (possibly, multiple times) to the node (or takes over a fraction of outbound connections of the node). In a low-fanout model, this attack is not feasible, because transactions are flooded

through outbound connections only.

In-flight collisions are also not possible in the case of low-fanout relay through only outbound links, because transactions are always announced in the same direction of a link.

Considering these arguments and to reduce latency, Erlay has a lower random diffusion interval. Instead of using $T_{oi} = 2$ seconds for outbound connections and $T_{ii} = 5$ seconds for inbound, Erlay uses $T_{oi} = 1$ seconds for outbound.

Reconciliation diffusion delay. Even though in Erlay timing attacks by observing low-fanout flooding are not feasible, an attacker would be able to perform them through reconciliations. To make timing attacks through reconciliations more expensive to perform, we enforce every peer to respond to reconciliation requests after a small random delay (in our implementation, it is a Poisson-distributed random variable which is on average $T_{ri} = 3$ seconds) and rate-limit reconciliations per peer. This measure would make Erlay better than BTCFlood in terms of withstanding timing attacks.

Our measure in Erlay has the same idea as in flooding/low-fanout diffusion, however, having the ratio T_{ii}/T_{oi} higher makes timing attacks less accurate, because during T_{ii} (the average time before an attacker receives a transaction) a transaction would be propagated to more nodes in the network.

We chose the interval of 3 seconds because a lower interval would make Erlay more susceptible to timing attacks than Bitcoin, and a higher interval results in a high latency.

Chapter 7

Evaluation methodology

In evaluating Erlay we focus on answering the following three questions:

1. How does Erlay compare in latency (the time that it takes for the transaction to reach all of the nodes) and bandwidth (number of bits used to disseminate a transaction) to BTCFlood?

2. How do the two parts of Erlay (low-fanout flooding and reconciliation) perform at scale and with varying number of nodes?

3. How do malicious nodes impact Erlay's performance?

We use measurement results from two sources to answer the questions above. First, we use a simulator to simulate Erlay on a single machine. Second, we implemented Erlay in the mainline Bitcoin client and deployed a network of Erlay clients on the Azure cloud across several data centers (Section 9).

Simulator design. Our simulation was done with ns3. We modified an open-source Bitcoin Simulator [28] to support transaction relay. The original simulator had 9663 LOC; we modified 9948 LOC.

Our simulator is based on the INV-GETDATA transaction relay protocol (see Section 2). It is parameterized by the current ratio of public nodes to private nodes in the Bitcoin network and the transaction rate based on the historical data from the Bitcoin network (on average, 7 transactions per second). To simulate the dif-

ferent ratios of faults in the network by introducing Black Hole nodes (nodes which receive messages, but do not propagate transactions further).

However, our simulator does not account for the resource-wise heterogeneous setting, the block relay phase, the joining and leaving of nodes during the transaction relay phase (churn), and we do not consider sophisticated malicious nodes.

The propagation latency measured for BTCFlood by our simulator matches the value suggested for the validation of Bitcoin simulators [22], and our measured bandwidth matches our analytical estimate.

Topology of the simulated network. We emulated a network, similar to the current Bitcoin network, since inferring the Bitcoin network topology is non-trivial [50]. In our simulation we bootstrap the network in 2 phrases: (1) public nodes connect to each other using a limit of 8 outbound connections, then (2) private nodes connect to 8 random public nodes.

Unless stated otherwise, our simulation results are for a network of 6,000 public nodes and 60,000 private nodes (this is the scale of today's network[1]). In each experiment we first used the above two steps to create the topology, then we relayed transactions for 600 seconds (on average, 4,200 transactions generated from random private nodes).

[1] https://bitnodes.earn.com/ https://luke.dashjr.org/programs/bitcoin/files/charts/software.html

Chapter 8

Simulation results

In this section we demonstrate the measured properties (latency, bandwidth consumption, and security) of Erlay and compare these to BTCFlood.

8.0.1 Relay bandwidth usage

To verify that Erlay scales better than BTCFlood as the number of connections per node is increased, we varied the number of outbound connections per node and measured the bandwidth used for relaying transactions.

We report the average relay bandwidth cost relatively to b (the size of a transaction announcement in bits). Table 8.1 shows the results. In the table, both BTCFlood and Erlay numbers are from simulation experiments.

Erlay achieves a cost below b for connectivity of 8 because of the use of short IDs during set reconciliation. This is because BTCFlood announces transactions on *every* link in the network, so its relay bandwidth increases linearly with the connectivity. By contrast, with higher connectivity of the network, bandwidth in Erlay grows significantly slower than in BTCFlood. This higher connectivity allows for better security of the overall system.

Transaction announcements in overall bandwidth. To demonstrate that Erlay's announcement optimization impacts overall bandwidth, we measure the bandwidth consumed by a simulated network to relay transactions with BTCFlood and Erlay. Fig. 8.1 plots the results for simulations in which every node establishes 8 connections. Erlays's announcement bandwidth is just 12.5% of the relay band-

Table 8.1: The average bandwidth cost of relaying one transaction to one node in the Bitcoin network with BTCFlood and Erlay. b is the announcement size in bits.

Connectivity	BTCFlood	Erlay
8	$8b$	$0.93b$
16	$16b$	$1.12b$
24	$24b$	$1.23b$
32	$32b$	$1.36b$

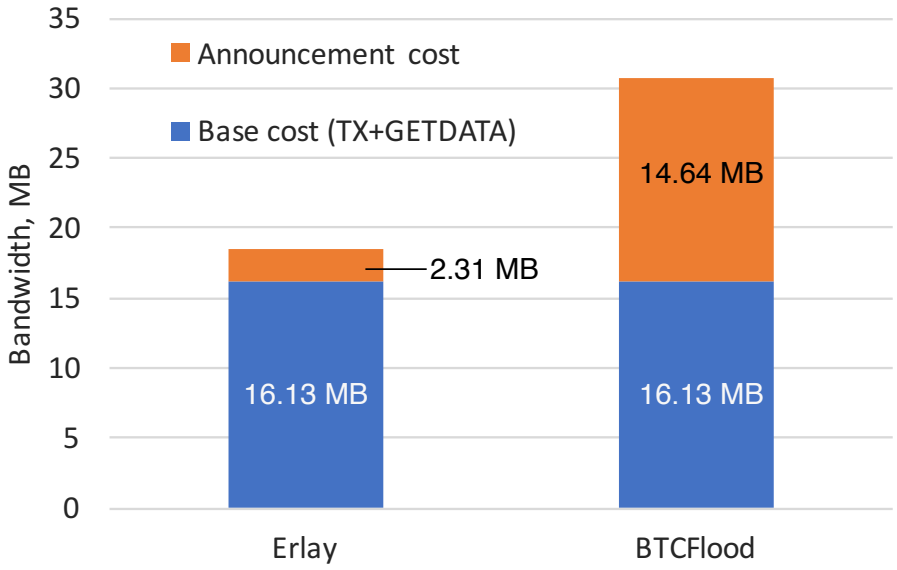

Figure 8.1: Average bandwidth cost of relaying a transaction in a network of 60,000 nodes with outbound connectivity of 8.

width, while for BTCFlood the announcement bandwidth is 44.7%.

Breaking down Erlay's bandwidth usage. To further understand Erlay's bandwidth usage, we broke it down by the different parts of the protocol: low-fanout flooding, reconciliation, post-reconciliation announcements.

Table 8.2 lists the results. The table shows that about a third of the bandwidth is used by low-fanout flooding, while reconciliation accounts for the bulk of the bandwidth. The post-reconciliation INVs account for a small fraction of Erlay's bandwidth.

Table 8.2: Breakdown of bandwidth usage in Erlay.

Erlay component	Bandwidth %
Low-fanout flooding	54%
Reconciliation	32%
Bisection	0.7%
Fallback	4.3%
Post-reconcile. INVs	9%
Total	**100%**

Set reconciliation effectiveness. To understand the effectiveness of Erlay's set reconciliation, we measured how often reconciliation or the following bisection protocol fail. Fig. 8.2 reports the results aggregated from one of our simulation runs with 60,000 nodes. The end-to-end probability of reaching fallback is below 1%. Since bisection does not introduce additional bandwidth overhead (while fallback does), the overall reconciliation overhead is low.

Since every reconciliation round requires a set difference estimation, we measured the distribution of the estimated difference sizes. Fig. 8.3 demonstrates that set difference depends on transaction rate. This is expected: for the same reconciliation intervals, a higher transaction rate would result in both reconciling parties receiving more transactions and would lead to a larger set difference. This dependency between set difference and transaction rate allows accurate set difference estimation. Fig. 8.2 illustrates that Erlay's estimate is correct 96% of the time. For the cases where Erlay over-estimates and the initial reconciliation fails, the resulting bandwidth overhead constinutes 9% of the overall bandwidth.

In our library benchmarks the decode time for a sketch containing 100 differences is under 1 millisecond (Fig. 6.1). Thus, the computational cost of operating over sketches with the distribution in Fig. 8.3 is negligible.

8.0.2 Relay latency

To analyze latency we ran simulation experiments while changing the total number of both private and public nodes. In these experiments we kept constant the ratio between private and public types of nodes at $10:1$ (this is the ratio in today's Bitcoin network). Fig. 8.4 plots the average latency for a single transaction to

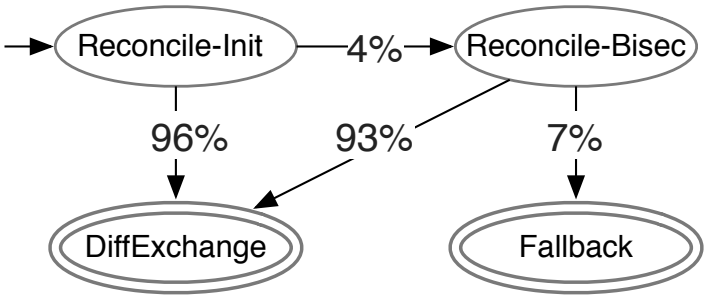

Figure 8.2: Finite state machine of the protocol in Fig. 2.2 annotated with transition percentages observed in our experiments.

reach all nodes for Erlay and BTCFlood. Erlay has a constant latency overhead on top of BTCFlood that is due to its use of batching. However, this overhead is just 2 seconds and changes at approximately the same rate with the number of nodes as BTCFlood's latency. We could further reduce Erlay's per transaction latency, but at the cost of higher bandwidth usage.

Latency under faulty condition We also evaluated Erlay's latency in a simple adversarial setting. For this we simulated a network in which 10% of the public nodes are *black holes* and measured the time for a transaction to reach all the nodes. While it is difficult to outperform the robustness of BTCFlood, an alternative protocol should not be dramatically impacted by this attack.

According to our measurements, while the slowdown with BTCFlood in this setting is 2%, the slowdown with Erlay is 20%. We believe that this latency increase is acceptable for a batching-based protocol. We have ideas for heuristics that might be applied to mitigate black-hole attacks and make Erlay less susceptible. For example, a node might avoid reconciling with those outbound connections that regularly provide the fewest new transactions.

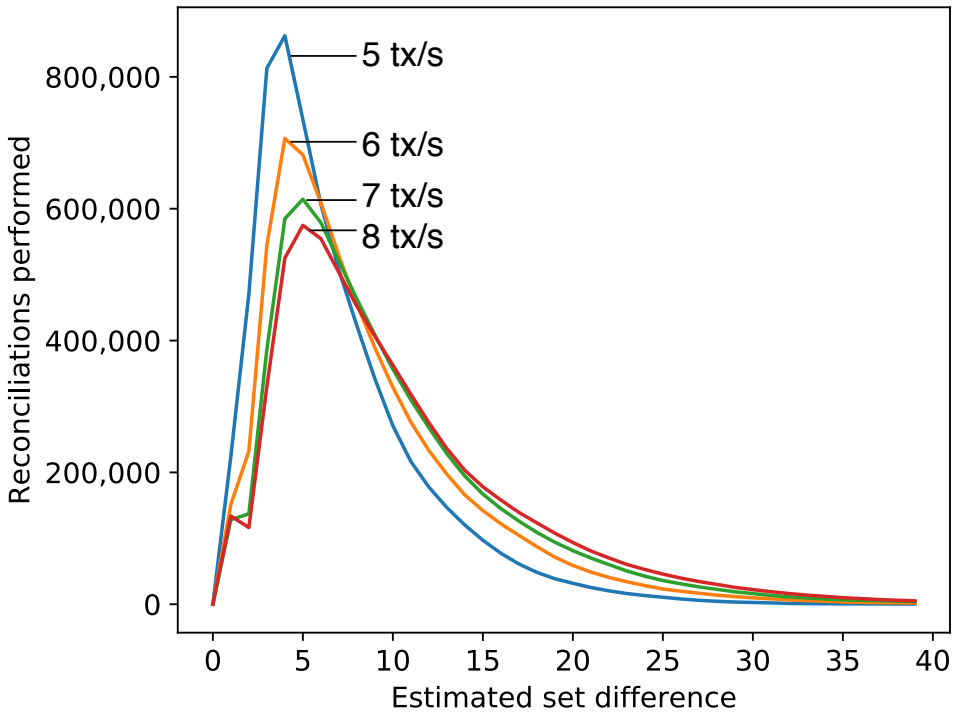

Figure 8.3: Distribution of the set difference estimates during reconciliation for different transaction rates.

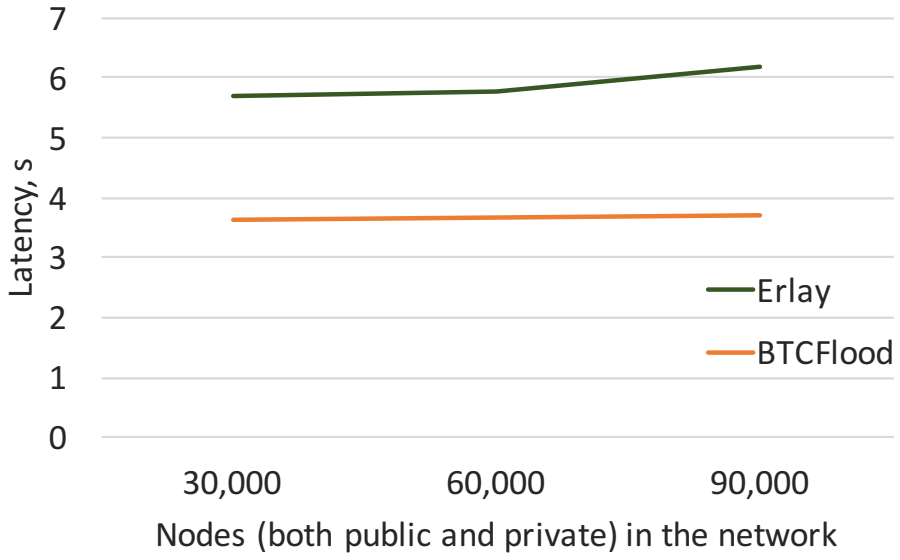

Figure 8.4: Average latency for a single transaction to reach 100% nodes in the network.

Chapter 9

Reference implementation results

We implemented Erlay as part of Bitcoin Core. For this we added 584 LOC, not including our PinSketch library. We used a network of 100 Azure nodes located in one data center, running a reference implementation of our protocol integrated in Bitcoin Core node software, to evaluate Erlay in deployment. We generated and relayed 500 transactions, all originating from one node with a rate of 7 transactions per second. We compared the average latency and bandwidth of Erlay versus Bitcoin's current implementation. Table 9.1 summarizes our results. According to our measurements, Erlay introduced a latency increase of 2.66 seconds, while saving 44% of the overall node bandwidth.

As in our simulations, Erlay has a higher latency but lower bandwidth cost, confirming our original design intent (Fig. 5.1).

	BTCFlood	Erlay
Base cost (MB) (TX+GETDATA)	12.78	12.75
Other messages (MB)	1.06	1.1
Announcement cost (MB)	17.79	2.70
Latency (s)	1.34	4.46

Table 9.1: Prototype measurements collected from a 100-node deployment comparing the latency and bandwidth of the BTCFlood in the reference implementation against our Erlay implementation.

Chapter 10

Discussion

Reconciliation-only relay. We believe that a reconciliation-only transaction relay protocol would be inherently susceptible to timing attacks that could reveal the source of the transaction. Unlike flooding, in reconciliation delays cannot be applied per-direction but rather per-link. Therefore, BTCFlood's diffusion delay cannot be used in reconciliation.

Compatibility with Dandelion. Dandelion is an alternative transaction relay protocol introduced to improve the anonymity and robustness to adversarial observers in Bitcoin [23]. Erlay is complimentary with Dandelion: Erlay would replace the fluff phase in Dandelion.

Backward compatibility. Only about 30% of Bitcoin nodes run the latest release of Bitcoin Core[1]. Therefore, Erlay must be backwards compatible. If not all the nodes use Erlay, then Erlay may be activated per-link if both peers support it.

Timing attacks from public nodes. Erlay's design is more robust to timing attacks from *private* sybils [18, 29]. Robustness to timing attacks from *public* nodes remains an open questions. Both BTCFlood and Erlay use faster relay to public nodes and slower relay to private nodes. But Erlay's slower reconciliation could be abused by public nodes to more easily identify transactions originating from private nodes. With higher node connectivity this attack becomes expensive, but a general solution remains as future work.

[1] https://luke.dashjr.org/programs/bitcoin/files/charts/security.html

Chapter 11

Related Work

Prior studies of Bitcoin's transaction relay focused on information leakage and other vulnerabilities [23, 50], and did not consider bandwidth optimization. We believe that our work is the first to introduce a bandwidth-efficient, low-latency, and robust transaction relay alternative for Bitcoin. Erlay is designed as a minimal change to Bitcoin (584 LOC), in contrast with other proposals that optimize Bitcoin more deeply [21].

Short transaction identifiers. One solution to BTCFlood's inefficiency is to use *short transaction identifiers*. There are two issues with this solution. First, this *only reduces bandwidth cost by a constant factor*. In our simulation we found that short identifiers would reduce redundant traffic from 43% to 10%. But, with higher connectivity, redundancy climbs back up faster than in does with Erlay. The second issue with short IDs is that they would make the system vulnerable to collision-related attacks and requires a new per-node or per-link secure salting strategy.

Blocksonly setting. Bitcoin Core 0.12 introduced a *blocksonly* setting using which a node does not send or receive individual transactions; instead, the node only handles complete blocks. As a result, blocksonly has no INV message overhead. In the blocksonly case, nodes will have to relay and receive many transactions at once. This will increase the maximum node bandwidth requirements and cause spikes in block content relay and transaction validation

Reconciliation alternatives. Prior work has also devised multi-party set reconcil-

iation [10, 46]. This approach, however, has additional complexity and additional trust requirements between peers. We believe that the benefits of such an approach are not substantial enough to justify these limitations.

In addition, reconciliation-based techniques usually provide bandwidth-efficiency under the assumptions where most of the state being reconciled is shared [13, 51].

Prior P2P research. Structured P2P networks are usually based on a Distributed Hash Tables (DHTs), in which every peer is responsible for specific content [41]. In these networks research has explored the use of topology information to make efficient routing decisions [12, 54, 56, 59]. This design, however, makes these protocols leak information about the structure of the network and makes them less robust to Byzantine faults; though *limited* solutions to Byzantine faults in this setting have been explored [15, 25].

The trade-off between latency and bandwidth efficiency is well-known in P2P research. Kumar et. al. identified and formalized the trade-off between latency and bandwidth [36], and Jiang et. al. proposed a solution to achieve an optimal combination of these properties [31]. However, the solution was not designed for adversarial settings.

Prior work also proposed feedback-based approaches to flooding [2, 52]. However, we believe that to work efficiently (have a *horizon* larger than 1), this work would have unacceptable information leakage.

Chapter 12

Conclusion

Bitcoin is one of the most widely used P2P applications. Today, Bitcoin relies on flooding to relay transactions in a network of about 60,000 nodes. Flooding provides low latency and is robust to adversarial behavior, but it is also bandwidth-inefficient and creates a significant amount of redundant traffic. We proposed Erlay, an alternative protocol that combines limited flooding with intermittent reconciliation. We evaluated Erlay in simulation and with a practical deployment. Compared to Bitcoin's current protocols, Erlay reduces the bandwidth used to announce transactions by 84% while increasing the latency for transaction dissemination by 50% (from 4s to 6s). Erlay allows Bitcoin nodes to have higher connectivity, which will make the network more secure. We are actively working to introduce Erlay into Bitcoin Core's node software.

Bibliography

[1] Bitcoin core commit 5400ef. https://github.com/bitcoin/bitcoin/commit/5400ef6bcb9d243b2b21697775aa6491115420f3. → page 4

[2] W. Ai, L. Xinsong, and L. Kejian. Efficient flooding in peer-to-peer networks. 01 2006. doi:10.1109/CAIDCD.2006.329410. → page 36

[3] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002. doi:10.1103/RevModPhys.74.47. URL https://link.aps.org/doi/10.1103/RevModPhys.74.47. → page 8

[4] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378 EP –, 07 2000. URL https://doi.org/10.1038/35019019. → page 8

[5] G. Andresen. Increase maximum block size. https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki, Jun 2015. → page 1

[6] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek. On the malleability of bitcoin transactions. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *Financial Cryptography and Data Security*, pages 1–18, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48051-9. → page 1

[7] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of useful work. 2017. → page 1

[8] A. Biryukov, D. Khovratovich, and I. Pustogarov. Deanonymisation of Clients in Bitcoin P2P Network. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2014. → pages 1, 8

[9] J. Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In *Financial Cryptography Workshops*, 2016. → page 1

[10] A. Boral and M. Mitzenmacher. Multi-party set reconciliation using characteristic polynomials. *CoRR*, abs/1410.2645, 2014. → page 36

[11] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8132-2. URL http://dl.acm.org/citation.cfm?id=829502.830043. → page 17

[12] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, pages 46–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-44702-3. doi:10.1007/3-540-44702-4_4. URL https://doi.org/10.1007/3-540-44702-4_4. → page 36

[13] M. Corallo. Bip 152: Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, 2016. → pages 1, 10, 36

[14] corbixgwelt. Timejacking and bitcoin. http://culubas.blogspot.de/2011/05/timejacking-bitcoin802.html, 2011. → page 1

[15] A. Dearle, G. N. C. Kirby, and S. J. Norcross. Hosting byzantine fault tolerant services on a chord ring. *CoRR*, abs/1006.3465, 2010. → page 36

[16] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013. doi:10.1109/P2P.2013.6688704. → page 8

[17] C. Decker and R. Wattenhofer. Bitcoin transaction malleability and mtgox. *CoRR*, abs/1403.6676, 2014. → page 1

[18] S. Delgado Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions, 12 2018. → pages 1, 8, 15, 34

[19] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages

523–540, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24676-3. → pages 15, 21

[20] J. R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems*, IPTPS, 2002. → page 1

[21] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *Usenix Conference on Networked Systems Design and Implementation*, NSDI, 2016. → page 35

[22] M. Fadhil, G. Owen, and M. Adda. Bitcoin network measurements for simulation validation and parameterisation. In *Proceedings of the Eleventh International Network Conference (INC 2016)*, pages 109–114. University of Plymouth, 7 2016. → page 25

[23] G. C. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *CoRR*, abs/1805.11060, 2018. URL http://arxiv.org/abs/1805.11060. → pages 34, 35

[24] J. Feigenbaum, S. Kannan, M. J. Strauss, and M. Viswanathan. An approximate l1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, Jan. 2003. ISSN 0097-5397. doi:10.1137/S0097539799361701. URL https://doi.org/10.1137/S0097539799361701. → page 17

[25] A. Fiat, J. Saia, and M. Young. Making chord robust to byzantine attacks. In *Proceedings of the 13th Annual European Conference on Algorithms*, ESA'05, pages 803–814, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-29118-0, 978-3-540-29118-3. doi:10.1007/11561071_71. URL http://dx.doi.org/10.1007/11561071_71. → page 36

[26] J. Garzik. Block size increase to 2mb. https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki, Jun 2015. → page 1

[27] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun. Tampering with the Delivery of Blocks and Transactions in Bitcoin. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2015. → page 1

[28] A. Gervais, G. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In

Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communication Security (CCS). ACM, 2016. → page 24

[29] M. Grundmann, T. Neudecker, and H. Hartenstein. Exploiting transaction accumulation and double spends for topology inference in bitcoin. 2018. → pages 8, 34

[30] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse Attacks on Bitcoin's Peer-to-peer Network. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC, 2015. → pages 1, 8

[31] J. Jiang, C. Hung, and J. Wu. Bandwidth- and latency-aware peer-to-peer instant friendcast for online social networks. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pages 829–834, Dec 2010. doi:10.1109/ICPADS.2010.101. → page 36

[32] jl777. Re: Blocksonly mode bw savings, the limits of efficient block xfer, and better relay. https://bitcointalk.org/index.php?topic=1377345.msg14012877#msg14012877, Feb 2016. → page 2

[33] B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore. Game-theoretic analysis of ddos attacks against bitcoin mining pools. In R. Böhme, M. Brenner, T. Moore, and M. Smith, editors, *Financial Cryptography and Data Security*, pages 72–86, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44774-1. → page 1

[34] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer. bloxroute: A scalable trustless blockchain distribution network whitepaper. → page 1

[35] P. Koshy, D. Koshy, and P. McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In N. Christin and R. Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 469–485, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-45472-5. → page 1

[36] P. Kumar, G. Sridhar, and V. Sridhar. Bandwidth and latency model for dht based peer-to-peer networks under variable churn. In *2005 Systems Communications (ICW'05, ICHSN'05, ICMCS'05, SENET'05)*, pages 320–325, Aug 2005. doi:10.1109/ICW.2005.31. → page 36

[37] E. Lombrozo, J. Lau, and P. Wuille. Segregated witness (consensus layer). https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, Dec 2015. → page 1

[38] S. V. Margariti and V. V. Dimakopoulos. A study on the redundancy of flooding in unstructured p2p networks. *International Journal of Parallel, Emergent and Distributed Systems*, 28(3):214–229, 2013. doi:10.1080/17445760.2012.724067. → page 2

[39] A. Maria, Z. Aviv, and V. Laurent. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP)*. IEEE, 2017. → pages 1, 8, 10

[40] G. Maxwell. Blocksonly mode bw savings, the limits of efficient block xfer, and better relay. https://bitcointalk.org/index.php?topic=1377345.0, Feb 2016. → page 2

[41] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Peer-to-Peer Systems*, pages 53–65, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45748-0. → page 36

[42] P. McCorry, S. F. Shahandashti, and F. Hao. Refund attacks on bitcoin's payment protocol. *IACR Cryptology ePrint Archive*, 2016:24, 2016. → page 1

[43] A. Miller. Feather-forks: enforcing a blacklist with sub-50% hashpower? https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391, 2013.

[44] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee. Discovering bitcoin ' s public topology and influential nodes. 2015. → pages 1, 8

[45] Y. Minsky and A. Trachtenberg. Practical Set Reconciliation. Dept. Elec. Comput. Eng., Boston Univ., Boston, MA, Tech. Rep. BU-ECE2002-01, 2002. → pages 3, 13

[46] M. Mitzenmacher and R. Pagh. Simple multi-party set reconciliation. *CoRR*, abs/1311.2037, 2013. → page 36

[47] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf. → page 1

[48] A. Narayanan. *Bitcoin and cryptocurrency technologies : a comprehensive introduction*. Princeton University Press, Princeton, New Jersey, 2016. ISBN 978-0691171692. → page 1

[49] T. Neudecker and H. Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys Tutorials*, pages 1–1, 2018. ISSN 1553-877X. doi:10.1109/COMST.2018.2852480. → page 8

[50] T. Neudecker, P. Andelfinger, and H. Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 358–367, July 2016. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070. → pages 1, 4, 8, 11, 15, 22, 25, 35

[51] A. P. Ozisik, G. Andresen, G. Bissias, A. Houmansadr, and B. N. Levine. Graphene: A new protocol for block propagation using set reconciliation. In *DPM/CBT@ESORICS*, volume 10436 of *Lecture Notes in Computer Science*, pages 420–428. Springer, 2017. → pages 1, 36

[52] C. Papadakis, P. Fragopoulou, E. P. Markatos, E. Athanasopoulos, M. Dikaiakos, and A. Labrinidis. *A Feedback-Based Approach to Reduce Duplicate Messages in Unstructured Peer-To-Peer Networks*, pages 103–118. Springer US, Boston, MA, 2007. ISBN 978-0-387-47658-2. doi:10.1007/978-0-387-47658-2_8. URL https://doi.org/10.1007/978-0-387-47658-2_8. → page 36

[53] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016. → page 1

[54] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001. → page 36

[55] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1): 42–81, Mar. 2005. ISSN 0360-0300. doi:10.1145/1057977.1057980. URL http://doi.acm.org/10.1145/1057977.1057980. → page 14

[56] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001. → page 36

[57] P. Todd. Why scaling bitcoin with sharding is very hard. https://petertodd.org/2015/why-scaling-bitcoin-with-sharding-is-very-hard, 2015. → page 2

[58] A. Trachtenberg, D. Starobinski, and S. Agarwal. Fast PDA synchronization using characteristic polynomial interpolation. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1510–1519 vol.3, June 2002. doi:10.1109/INFCOM.2002.1019402. → page 21

[59] S. Vuong and J. Li. Efa: an efficient content routing algorithm in large peer-to-peer overlay networks. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P)*, pages 216–217, Sept 2003. doi:10.1109/PTP.2003.1231532. → page 36

[60] P. Wuille. Block size following technological growth. https://github.com/bitcoin/bips/blob/master/bip-0103.mediawiki, Jul 2015. → page 1

[61] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. V. Renesse. REM: Resource-Efficient Mining for Blockchains. In *USENIX Security Symposium (USENIX Security)*, 2017. → page 1