Scaling Lightning Safely With Feerate-Dependent Timelocks

John Law

December 15, 2023

Version 1.1

Abstract

All known Lightning channel and factory protocols are susceptible to forced expiration spam attacks, in which an attacker floods the blockchain with transactions in order to prevent honest users from putting their transactions onchain before timelocks expire. This paper proposes a change to Bitcoin's consensus rules that supports Feerate-Dependent Timelocks (FDTs). FDTs are timelocks that automatically extend when the feerate for putting transactions onchain spikes (such as would occur during a forced expiration spam attack). In the normal case, there is no spike in the feerate and thus no tradeoff between capital efficiency and safety. However, if a dishonest user attempts a forced expiration spam attack, feerates increase and FDTs are extended, thus penalizing the attacker by keeping their capital timelocked for longer. FDTs are tunable and can be made to be highly resistant to attacks from dishonest miners. In addition to mitigating forced expiration spam attacks, FDTs can reduce the risk of having to pay unexpectedly high fees during a congestion spike and they can improve the accuracy of fee-penalties.

Of separate interest, an exact analysis of the risk of double spend attacks is presented that corrects an error in the original Bitcoin whitepaper.

1 Overview

Because the Lightning protocol relies on timelocks to establish the correct channel state, Lightning users could lose their funds if they are unable to put their transactions onchain quickly enough **[BOLT]**. The original Lightning paper **[PD16]** states that "[f]orced expiration of many transactions may be the greatest systemic risk when using the Lightning Network" and it uses the term "forced expiration spam" for an attack in which a malicious party "creates many channels and forces them all to expire at once", thus allowing timelocked transactions to become valid. That paper also says that the creation of a credible threat against "spamming the blockchain to encourage transactions to timeout" is "imperative" **[PD16]**.

Channel factories that create multiple Lightning channels with a single onchain transaction **[BDW18][DRO18][Law22d][Law23a]** increase this risk in two ways. First, factories allow more channels to be created, thus increasing the potential for many channels to require onchain transactions at the same time. Second, channel factories themselves use timelocks, and thus are vulnerable to a "forced expiration spam" attack.

In fact, the timelocks in Lightning channels and factories are risky even without an attack from a malicious party. Blockchain congestion is highly variable and new applications (such as ordinals) can cause a sudden spike in congestion at any time. As a result, timelocks that were set when congestion was low can be too short when congestion spikes. Even worse, a spike in congestion could be self-reinforcing if it causes malicious parties to attack opportunistically and honest parties to put their channels onchain due to the heightened risk.

One way to reduce the risk of a forced expiration spam attack is to use longer timelocks that give honest users more time to put their transactions onchain. However, long timelocks limit the ability to dynamically reassign the channel's (or factory's) funds, thus creating a tradeoff between capital efficiency and safety **[Tow23]**. While long timelocks could maintain safety for small numbers of channels, supporting billions (or tens of billions) of channels while maintaining safety is probably impossible **[Law23b]**.

Another way to reduce risk is to impose a penalty on an attacker. Some channel protocols, such as the original Lightning protocol **[PD16]**, a version of the two-party eltoo protocol **[Tow22]**, the fully-factory-optimized protocol **[Law22c]**, and the tunable-penalty channel protocol **[Law22b]** include such penalties. In addition, the tunable-penalty and single-commitment factory protocols **[Law22d]** support penalties. However, as was noted in the original Lightning paper **[PD16]**, penalties do not eliminate the risk of a forced expiration spam attack. Furthermore, existing penalty-based factory protocols **[Law22d]** have limited scalability, as they depend on getting large numbers of casual users to coordinate and co-sign transactions **[Ria23][Law23a]**.

In contrast, the timeout-tree protocol **[Law23a]** scales via simple covenants (enabled by support for CheckTemplateVerify, AnyPrevOut, or a similar change to the Bitcoin consensus rules). As a result, a single timeout-tree can support millions of channels and one small transaction per block can fund timeout-trees with tens of billions of offchain channels **[Law23a]**. However, timeout-trees do not support penalties, and like all other known factory protocols **[BDW18][DRO18][Law22d]**, timeout-trees rely on timelocks.

Therefore, if the need to protect against forced expiration spam was already "imperative" for the original Lightning channel protocol **[PD16]**, the use of scalable channel factories will make such protection indispensable.

This paper proposes a change to Bitcoin's consensus rules that allows the length of a timelock to depend on the feerate being charged for putting transactions onchain. Such Feerate-Dependent

Timelocks (FDTs) can be used to make the above channel and factory protocols resistant to sudden spikes in blockchain congestion. In the normal case, when there is no spike in congestion, FDTs do not extend the lengths of timelocks and thus do not create a tradeoff between capital efficiency and safety. On the other hand, when congestion spikes, FDTs extend the lengths of timelocks and thus penalize the owner of the timelocked capital by reducing its efficiency. Therefore, FDTs can be viewed as creating a penalty for spamming the blockchain, thus reducing the likelihood of such an attack even if the channel (or factory) protocol being used does not have an explicit penalty mechanism. FDTs have other uses, including reducing the risk of having to pay unexpectedly high fees during a congestion spike, improving the accuracy of fee-penalties **[Law23a]** and reducing the risk of one-shot receives **[Law22a]**.

Of separate interest, the analysis of FDTs given here leads to an exact analysis of the risk of double spend attacks that corrects an error in the original Bitcoin whitepaper **[Nak09]**.

2 Feerate-Dependent Timelock (FDT) Proposal

A Feerate-Dependent Timelock (FDT) is created by encoding a feerate upper bound in a transaction's nSequence field. A transaction with an FDT cannot be put onchain until:

1. its absolute timelock encoded in its nLocktime field (and its relative timelock encoded in the same nSequence field, if present) has been satisfied, and
2. the prevailing feerate has fallen below the FDT's feerate upper bound.

As a result, FDTs are automatically extended when the feerate for putting transactions onchain spikes (such as would occur during a forced expiration spam attack).

In order to determine the prevailing feerate, the median feerate of each block is calculated as the feerate in satoshis per virtual byte (vbyte) that is paid for at least half of the block's vbytes.

If all miners were honest, a single block with a low median feerate would be enough to guarantee that congestion is low. However, even a small fraction of dishonest miners would be able to occasionally mine a block with an artificially low feerate. As a result, it is not safe to wait for one block (or some other fixed number of blocks) with a low feerate in order to guarantee that honest users can put their transactions onchain.

Instead, an FDT requires that some maximum number of blocks within an aligned window of consecutive blocks have a high median feerate. The FDT proposal uses 14 currently masked-off bits in the nSequence field to express the FDT's three parameters:

1. *feerate_value*,
2. *window_size*, and
3. *block_count*.

An aligned window of *window_size* blocks satisfies the FDT's parameters if it has fewer than *block_count* blocks with median feerate above *feerate_value*. A transaction with an FDT can only be put onchain after an aligned window that satisfies the FDT's parameters and starts no earlier than when the transaction's absolute timelock (and corresponding relative timelock, if present) is satisfied.

In addition, the CheckSequenceVerify (CSV) operator is extended to enforce the desired *feerate_value*, *window_size* and *block_count*.

The specification of the FDT proposal is given in Appendix A.

3 Safe Lightning Channels And Factories

In order to protect a channel or factory protocol against forced expiration spam attacks, the protocol's timelocks are made to be feerate-dependent. This is done by selecting a *feerate_value* (such as 4 times the current feerate) that would be caused by a forced expiration spam attack, along with the desired *window_size* and *block_count* parameters.

It is also possible to create multiple conflicting transactions with different FDTs (with later timelocks allowing higher feerates) in order to avoid timelocks that will never expire if feerates remain high permanently. Multiple FDTs work well with timeout-trees **[Law23a]** and in protocols (such as eltoo **[DRO18][Tow22]**) that use AnyPrevOut, as such protocols do not require the offline signing of separate transactions for each FDT. On the other hand, the current Lightning channel protocol **[BOLT]** and the Tunable-Penalty channel **[Law22b]** and factory **[Law22d]** protocols do require creating separate signed transactions for each value of the Commitment transaction's FDT, thus limiting the number of timelocks that are useful in practice.

Finally, in addition to forced expiration spam attacks, there is the possibility that a dishonest party will create *block_count* blocks with a high feerate in each window in order to artificially delay an FDT. The dishonest party does not directly benefit from such a delay, and they can be made to pay a higher cost for maintaining the delay by selecting a larger value for the *block_count* parameter.

4 Other Uses

FDTs have uses in addition to protecting channel and factory protocols from forced expiration spam attacks.

First, FDTs can protect users that are racing against timelocks from having to pay an unexpectedly high feerate due to temporary feerate fluctuations. For example, assume that B is a dedicated user who creates a timeout-tree where each leaf *i* funds a Lightning channel owned by casual user A_i and B. The timeout-tree can have an initial expiry that uses an FDT with a relatively low feerate (such as 1.25 times the current feerate) and a 3-months-later expiry that uses an FDT with a higher feerate (such as 4 times the current feerate). In this case, casual user A_i can attempt to put their leaf onchain using a low feerate (e.g., 1.5 times the current feerate), and if this fails due to its feerate, A_i can try again 3 months later using a higher feerate (e.g., 5 times the current feerate).

Next, FDTs can be used to improve the accuracy of fee-penalties that are assessed when a casual user puts their timeout-tree leaf onchain (Section 4.10 of **[Law23a]**). A fee-penalty is an amount that a casual user pays to the funder of the timeout-tree in order to reimburse the funder for fees imposed upon the funder by the casual user. When FDTs are not supported, the fee-penalty is a function of the number of transactions that the funder will have to put onchain (Section 4.10 of **[Law23a]**), but is independent of the feerates that prevail when the funder puts those transactions onchain. When FDTs are supported, the fee-penalty can be made to depend on the prevailing feerates, also. To do this, the timeout-tree leaf's fee-penalty output can be spent by one of several possible Feerate-Establishing (F-e) transactions that is put on-chain by the casual user. Each possible Feerate-Establishing transaction has an FDT with a unique feerate, an absolute timelock that allows it to be put onchain as soon as the casual user may have to put the leaf onchain, and no relative timelock. The casual user puts the Feerate-Establishing transaction with the lowest possible feerate onchain. That Feerate-Establishing transaction has an output that can be spent by a Fee-Splitting (F-s) transaction that is put onchain by the timeout-tree's funder. As a result, the fee-penalty amount can depend on both the number of transactions the funder must put onchain and the prevailing feerate.

Finally, FDTs can be used to allow a casual user to submit a transaction to the blockchain without having to then monitor the blockchain for a sudden spike in feerates, thus reducing the risk of one-shot receives **[Law22a]**. For example, the WF protocol allows a casual user to unilaterally receive a Lightning payment without having to monitor the blockchain for a conflicting transaction, as long as the casual user puts their Commitment and HTLC-success transactions onchain before their channel partner puts a conflicting transaction onchain **[Law22a]**. The channel partner's conflicting transaction has an absolute timelock, and by making that timelock feerate-dependent, the casual user can be assured that their Commitment has a sufficient feerate to win the race. Similarly, the FFO-WF protocol allows a casual user to put their HTLC-success transaction onchain before their channel partner can put their conflicting HTLC-timeout transaction onchain **[Law22c]**. The channel partner's HTLC-timeout transaction has an absolute timelock, and by making that timelock feerate-dependent, the casual user can be assured that their HTLC-success has a sufficient feerate to win the race. Of course, these protocols still depend on the forwarding of the casual user's transaction(s) to miners. Still, the use of FDTs protects the casual user from one form of risk (namely the risk of a spike in feerates).

5 Analysis

5.1 FDT Implementation Cost

In order to verify an FDT, nodes have to determine whether or not there is an aligned window with a sufficient number of low-feerate blocks after the FDT's absolute and relative timelocks were satisfied. Therefore, if a node knows the starting block of the most recent aligned window that satisfies the FDT's *feerate_value*, *window_size*, and *block_count* parameters, the node can compare that starting block with the FDT's timelocks to verify the FDT. Because the FDT parameters can be expressed using 14 bits, nodes only have to keep track of the starting block for 2^{14} = 16k different low-feerate windows. The starting block for each such window can be stored in 4 bytes, so 16k * 4B = 64kB of memory allows a node to verify an FDT in constant time. (In practice, slightly memory could be used in order to accommodate a reordering of the most recent 1k blocks.) Therefore, DRAM that costs less than one cent, plus a small constant number of computations, suffice to verify an FDT.

5.2 FDT Dishonest Miner Attacks

The *window_size* and *block_count* parameters can be selected to balance between:

1. latency,
2. the feerate paid by honest users, and
3. security against dishonest miners.

At one extreme, if dishonest miners are of no concern, *window_size* and *block_count* can be set to 1, so the FDT can be satisfied when the first block with a sufficiently low feerate is mined. At the other extreme, if dishonest miners are of great concern, *window_size* can be set to 16k and *block_count* can be set to 1024, in which case dishonest miners with 45% of the hashpower would have less than a 10^{-33} chance of dishonestly mining enough blocks in a given window to satisfy the FDT prior to the honest users being able to get their transactions onchain.

Exact formulas for the risk of FDT dishonest miner attacks, along with tables with some example parameters, are given in Appendix D. Programs that implement those formulas are available on GitHub **[Law23c]**.

5.3 Double Spend Attacks

While it is unrelated to FDTs, the analysis of FDTs' resistance to dishonest miner attacks can also be used to analyze the risk of double spend attacks. The original Bitcoin whitepaper **[Nak09]** includes an analysis of the probability of a double spend attack in which a dishonest party colludes with dishonest miners in order to undo a bitcoin transaction and steal the goods purchased with that transaction. That analysis correctly shows that the probability of success of a double spend attack falls exponentially with *z*, the depth of the transaction that is being double spent. However, there are two problems with that analysis:

1. it is approximate, and
2. it ignores the possibility of the dishonest miners using pre-mining.

The first problem was addressed by Grunspan and Perez-Marco **[GPM17]**. However, it does not appear that the second problem has been addressed previously.

Exact formulas for the risk of double spend attacks, including pre-mining, are given in Appendix C and programs that implement those formulas are available on GitHub **[Law23c]**.

The effect of including pre-mining only becomes apparent when a large fraction of the miners are dishonest. For example, Nakamoto estimates the required value of *z* to guarantee at most a 0.1% chance of a successful double spend, and Grunspan and Perez-Marco give exact values assuming no pre-mining. Those results, plus exact results with pre-mining, are as follows:

% dishonest miners	**Estimated *z* without pre-mining [Nak09]**	**Exact *z* without pre-mining [GPM17]**	**Exact *z* with pre-mining**
10	5	6	6
15	8	9	9
20	11	13	13
25	15	20	20
30	24	32	33
35	41	58	62
40	89	133	144
45	340	539	589

Table 1. Transaction depth (*z*) required to guarantee at most a 0.1% chance of a successful double spend attack, assuming the timing of the attack is not under the control of the attacker.

Examples of the probability of a successful double spend attack as estimated by Nakamoto, calculated by Grunspan and Perez-Marco assuming no pre-mining, and calculated with pre-mining, are given in the following two tables.

Transaction Depth	Estimated probability without pre-mining [Nak09]	Exact probability without pre-mining [GPM17]	Exact probability with pre-mining
0	1.0000000	1.0000000	1.0000000
1	0.2045873	0.2000000	0.2098765
2	0.0509779	0.0560000	0.0597531
3	0.0131722	0.0171200	0.0184296
4	0.0034552	0.0054560	0.0059062
5	0.0009137	0.0017818	0.0019363
6	0.0002428	0.0005914	0.0006445
7	0.0000647	0.0001986	0.0002169
8	0.0000173	0.0000673	0.0000736
9	0.0000046	0.0000229	0.0000251
10	0.0000012	0.0000079	0.0000086

Table 2. Probability of a successful double spend attack assuming 0.10 of the hashpower is dishonest and the timing of the attack is not under the control of the attacker.

Transaction Depth	Estimated probability without pre-mining [Nak09]	Exact probability without pre-mining [GPM17]	Exact probability with pre-mining
0	1.0000000	1.0000000	1.0000000
5	0.1773523	0.1976173	0.2463365
10	0.0416605	0.0651067	0.0845940
15	0.0101008	0.0233077	0.0309234
20	0.0024804	0.0086739	0.0116583
25	0.0006132	0.0033027	0.0044490
30	0.0001522	0.0012769	0.0017431
35	0.0000379	0.0004991	0.0006848
40	0.0000095	0.0001967	0.0002709
45	0.0000024	0.0000780	0.0001078
50	0.0000006	0.0000311	0.0000431

Table 3. Probability of a successful double spend attack assuming 0.30 of the hashpower is dishonest and the timing of the attack is not under the control of the attacker.

It is important to note that the above results with pre-mining assume that the time of the double spend attack is not selected by the attacker. If the attacker can select when to perform the attack, they are guaranteed to succeed given any value of z, but the expected time required to perform the attack grows exponentially with z.

6 Related Work

Poon and Dryja **[PD16]** identified the danger of a forced expiration spam attack on Lightning and highlighted the importance of preventing such attacks. Towns **[Tow23]** quantified the cost of stranded capital due to the need to extend timelocks in timeout-trees in order to tolerate forced expiration spam attacks.

Nakamoto **[Nak09]** estimated the probability of a successful double spend attack and Grunspan and Perez-Marco **[GPM17]** gave the exact probability for the success of such an attack. However, both of these works assume that the attacker is not able to pre-mine in order to get ahead of the public blockchain, when in fact such pre-mining is possible. The results given here differ by including the possibility of such pre-mining.

7 Conclusions

Securing Lightning channels and channel factories against forced expiration spam attacks is imperative.

Feerate-Dependent Timelocks (FDTs) provide this security without forcing the timelocks to be extended in the typical case, thus avoiding a capital efficiency vs. safety tradeoff. Furthermore, a dishonest user who tries to use a forced expiration spam attack to steal funds is penalized by having their funds timelocked for a longer period, thus discouraging such attacks. Finally, FDTs can be made to be highly resistant to attacks by dishonest miners.

FDTs have other uses, including the reduction of feerate risk and the calculation of fee-penalties.

While implementing FDTs requires some additional DRAM and computation, the costs are extremely small. Given these advantages and their low costs, it is hoped that the Bitcoin consensus rules will be changed to support FDTs.

References

BDW18 Conrad Burchert, Christian Decker and Roger Wattenhofer. Scalable Funding of Bitcoin Micropayment Channel Networks. In Royal Society Open Science, 20 July 2018. See http://dx.doi.org/10.1098/rsos.180089.

BOLT BOLT (Basis Of Lightning Technology) specifications. See https://github.com/lightningnetwork/lightning-rfc.

DRO18 Christian Decker, Rusty Russell and Olaoluwa Osuntokun. eltoo: A Simple Layer2 Protocol for Bitcoin. 2018. See https://blockstream.com/eltoo.pdf.

Fel57 William Feller. An Introduction To Probability Theory And Its Applications. 1957.

GPM17 Cyril Grunspan and Ricardo Perez-Marco. Double Spend Races. See CoRR, vol. abs/1702.02867 and https://arxiv.org/abs/1702.02867v3.

Law22a John Law. Watchtower-Free Lightning Channels For Casual Users. See https://github.com/JohnLaw2/ln-watchtower-free.

Law22b John Law. Lightning Channels With Tunable Penalties. See https://github.com/JohnLaw2/ln-tunable-penalties.

Law22c John Law. Factory-Optimized Channel Protocols For Lightning. See https://github.com/JohnLaw2/ln-factory-optimized

Law22d John Law. Efficient Factories For Lightning Channels. See https://github.com/JohnLaw2/ln-efficient-factories.

Law23a John Law. Scaling Lightning With Simple Covenants. See https://github.com/JohnLaw2/ln-scaling-covenants.

Law23b John Law. Re: Scaling Lightning With Simple Covenants. See https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2023-November/022175.html.

Law23c John Law. See https://github.com/JohnLaw2/ln-fdts.

PD16 Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments (Draft Version 0.5.9.2). See https://lightning.network/lightning-network-paper.pdf.

Nak09 Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. See http://bitcoin.org/bitcoin.pdf.

Ria23 Antoine Riard. Solving CoinPool High-Interactivity Issue With Cut-Through Update Of Taproot Leaves. See https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2023-September/021969.html.

Tow22 Anthony Towns. Two-Party eltoo w/ Punishment. See https://lists.linuxfoundation.org/pipermail/lightning-dev/2022-December/003788.html

Tow23 Anthony Towns. Re: Scaling Lightning With Simple Covenants. See https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2023-September/021943.html.

Appendix A: FDT Specification

FDTs, and OP_CHECKSEQUENCEVERIFY operations that enforce FDTs, are specified only for transactions with a sufficiently large nVersion (e.g., greater than or equal to 3).

In a transaction with a sufficiently large nVersion:

- nSequence[31] == 0: relative timelock enabled
    - nSequence[30..23] == 0x00: feerate-dependent timelock disabled
        - nSequence[22] == 0: relative timelock is expressed as number of blocks
        - nSequence[22] == 1: relative timelock is expressed as number of 512-seconds
        - nSequence[21..16]: masked off
        - nSequence[15..0]: *relative_locktime*
    - nSequence[30..23] > 0x00: feerate-dependent timelock enabled
        - nSequence[30..26]: *feerate_exponent*
        - nSequence[25..23]: *feerate_mantissa*
            - *feerate_value* = ((8 + *feerate_mantissa*) << *feerate_exponent*)
        - nSequence[22] == 0: relative timelock is expressed as number of blocks
        - nSequence[22] == 1: relative timelock is expressed as number of 512-seconds
        - nSequence[21..19]: *log_base_4_window_size*
            - *window_size* = (1 << (2 * *log_base_4_window_size*))
        - nSequence[18..16]: *log_base_4_block_count*
            - *block_count* = (1 << (2 * *log_base_4_block_count*))
        - nSequence[15..0]: *relative_locktime*
- nSequence[31] == 1: relative timelock disabled

If a transaction has an nSequence that enables a feerate-dependent timelock, that transaction can only be included in a block that follows a window W where:

- W is an aligned window of *window_size* consecutive blocks,
- W contains fewer than *block_count* blocks with median feerate greater than *feerate_value*,
- the first block in W satisfies the transaction's absolute locktime (nLocktime) constraint, and

- if *relative_locktime* > 0x00, the first block in W satisfies the transaction's relative locktime constraint.

The *feerate_value* is expressed in millisatoshis per virtual byte (vbyte).

Note that *feerate_value* can range from 0.009 satoshis/vbyte to (0xf << 31)/1000 ≈ 32,000,000 satoshis/vbyte, and that successive *feerate_value*s differ by a factor of at most 1.125.

If a transaction has a sufficiently large nVersion, OP_CHECKSEQUENCEVERIFY is redefined. When executed, if any of the following conditions are true, the script interpreter will terminate with an error:

- the stack is empty, or
- the top item on the stack is less than 0, or
- the top item on the stack has the disable flag (1 << 31) unset, and
    - the child transaction version is not sufficiently large to enable FDTs, or
    - the child transaction input sequence number disable flag (1 << 31) is set, or
    - the relative locktime types is not the same, or
    - the *relative_locktime* of the top stack item is greater than the child transaction input sequence number's *relative_locktime*, or
    - the top item on the stack specifies that feerate-dependent timelocks are enabled (that is, its bits 30 through 23 do not all equal 0) and
        - there exists a *b*, 16 ≤ *b* ≤ 30, such that bit (1 << *b*) of the top item of the stack does not equal bit (1 << *b*) of the child transaction input sequence number.

Appendix B: Random Walks

This appendix has three technical results regarding random walks that will be useful in analyzing risks of double spend and FDT dishonest miner attacks. The first lemma considers an unconstrained one-dimensional random walk while the other lemmas analyze a random walk with one reflecting barrier.

The first lemma was proven by Feller **[Fel57]** and was cited by Nakamoto **[Nak09]**.

Lemma 1 [Fel57]. Consider a particle that starts at 0 and moves +1 with probability $q < 0.5$ and -1 with probability $p = 1 - q$. Given any positive integer z, the probability that the particle will ever reach z is $(q/p)^z$.

Lemma 2. Consider a particle that starts at 0 and moves +1 with probability $q < 0.5$ and -1 with probability $p = 1 - q$ (unless it was at 0, in which case it stays at 0 with probability p). Given any integer $z \geq 0$, let S_z denote the steady state probability that the particle is at z. For any $z \geq 0$, $S_z = ((p - q)/p)(q/p)^z$.

Proof: First, note that for any $z \geq 0$ and $a \geq 0$, the particle is at z after exactly $z + a$ steps with probability at least $p^a q^z > 0$, so the position of the particle is aperiodic and steady state probabilities do exist.

For any $z \geq 0$, let $T_z = ((p - q)/p)(q/p)^z$. Note that S_z is the unique quantity that satisfies the three following equalities:

(1) $S_0 = pS_0 + pS_1$

(2) for any $z > 0$: $S_z = qS_{z-1} + pS_{z+1}$

(3) and $\sum_{z=0}^{\infty} S_z = 1.$

Therefore, if we can show that T_z satisfies those three equalities, we will have proven the lemma by showing that $T_z = S_z$. First, note that:

$$T_0 = (p - q)/p = (p + q)(p - q)/p = p(p - q)/p + q(p - q)/p = p(p - q)/p + p(q/p)(p - q)/p = pT_0 + pT_1 ,$$

so equality (1) holds for T_0. Next, note that for any $z > 0$:

$$T_z = ((p - q)/p)(q/p)^z = (p + q)((p - q)/p)(q/p)^z = p((p - q)/p)(q/p)^z + q((p - q)/p)(q/p)^z$$

$$= p(q/p)((p - q)/p)(q/p)^{z-1} + q(p/q)((p - q)/p)(q/p)^{z+1} = q((p - q)/p)(q/p)^{z-1} + p((p - q)/p)(q/p)^{z+1}$$

$$= qT_{z-1} + pT_{z+1}$$

so equality (2) holds for T_z . Finally, note that

$$\sum_{z=0}^{\infty} T_z = \sum_{z=0}^{\infty} ((p-q)/p)(q/p)^z$$

$$= ((p-q)/p) \sum_{z=0}^{\infty} (q/p)^z$$

$$= ((p-q)/p)/(1-(q/p))$$

$$= ((p-q)/p)/((p-q)/p)$$

$$= 1$$

so equality (3) holds for T_z and the proof is complete. □

Lemma 3. Consider a particle that starts at 0 and moves +1 with probability $q < 0.5$ and -1 with probability $p = 1 - q$ (unless it was at 0, in which case it stays at 0 with probability p). For any x and z, $0 \le x \le z$, let $E_{x,z}$ denote the expected number of steps that it takes for the particle to reach z from x and let $r = p/q$. For any x and z, $0 \le x \le z$: $E_{x,z}=(1/q)[(z-x)\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-1} ir^{x+i}]$.

Proof: For any x and z, $0 \le x \le z$, let $F_{x,z}=(1/q)[(z-x)\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-1} ir^{x+i}]$.

Note that $E_{x,z}$ is the unique quantity that satisfies the three following equalities:

(1) for any $z \ge 0$: $E_{z,z} = 0$

(2) for any $z > 0$: $E_{0,z} = 1 + pE_{0,z} + qE_{1,z}$

(3) and for any $0 < x < z$: $E_{x,z} = 1 + pE_{x-1,z} + qE_{x+1,z}$.

Therefore, if we can show that $F_{x,z}$ satisfies those three equalities, we will have proven the lemma by showing that $F_{x,z} = E_{x,z}$.

First, note that for any $z \ge 0$:

$$F_{z,z}=(1/q)[(z-z)\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-z-1} ir^{z+i}]=0,$$

so equality (1) holds for $F_{z,z}$.

Next, note that for any $0 \le x < z$:

$$\begin{aligned}F_{x,z}&=(1/q)[(z-x)\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-1} ir^{x+i}]\\ &=(1/q)[(z-x-1)\sum_{i=0}^{z-1} r^i + \sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-2} ir^{x+i+1} - \sum_{i=1}^{z-x-1} r^{x+i}]\\ &=(1/q)[(z-x-1)\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-2} ir^{x+i+1}]+(1/q)[\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-1} r^{x+i}]\\ &=F_{x+1,z}+(1/q)[\sum_{i=0}^{z-1} r^i - \sum_{i=1}^{z-x-1} r^{x+i}]\\ &=F_{x+1,z}+(1/q)\sum_{i=0}^{x} r^i\end{aligned}$$

so for any $0 \le x < z$:

(4) $$F_{x,z}=F_{x+1,z}+(1/q)\sum_{i=0}^{x} r^i.$$

Therefore, for any 0 < *x* ≤ *z*:

$$F_{x-1,z} = F_{x,z} + (1/q)\sum_{i=0}^{x-1} r^i$$

and for any 0 < *x* ≤ *z*:

(5) $$F_{x,z} = F_{x-1,z} - (1/q)\sum_{i=0}^{x-1} r^i.$$

Therefore, for any *z* > 0:

$$F_{0,z} = F_{1,z} + (1/q)\sum_{i=0}^{0} r^i = F_{1,z} + (1/q)r^0 = F_{1,z} + (1/q)$$

so

$$qF_{0,z} = qF_{1,z} + 1$$

and

$$pF_{0,z} + qF_{0,z} = pF_{0,z} + qF_{1,z} + 1$$

and

$$F_{0,z} = 1 + pF_{0,z} + qF_{1,z}$$

so equality (2) holds for $F_{0,z}$.

Finally, for any 0 < *x* < *z*, using (5) and (4) it follows that:

$$\begin{aligned}
F_{x,z} &= pF_{x,z} + qF_{x,z} \\
&= pF_{x-1,z} - (p/q)\sum_{i=0}^{x-1} r^i + qF_{x+1,z} + \sum_{i=0}^{x} r^i \\
&= pF_{x-1,z} + qF_{x+1,z} - r\sum_{i=0}^{x-1} r^i + \sum_{i=0}^{x} r^i \\
&= pF_{x-1,z} + qF_{x+1,z} - \sum_{i=1}^{x} r^i + \sum_{i=0}^{x} r^i \\
&= pF_{x-1,z} + qF_{x+1,z} + r^0 \\
&= 1 + pF_{x-1,z} + qF_{x+1,z}
\end{aligned}$$

so equality (3) holds for $F_{x,z}$ and the proof is complete. □

Appendix C: Risk Of Double Spend Attack

A *double spend attack* occurs when a dishonest party pays an honest party with a transaction T_h, receives goods or services from the honest party, and then puts another transaction T_d in the blockchain that pays to the dishonest party and double spends the same output spent by T_h, thus invalidating T_h. In order to get T_d in the final version of the blockchain, the dishonest party controls enough hashpower to create a secret branch of the blockchain that includes T_d rather than T_h and is as long as[1] the public branch containing T_h. The honest party tries to thwart the dishonest party by waiting until T_h is some number of blocks *z* deep in the public blockchain. The remainder of this appendix considers the probability of a successful double spend attack under three different scenarios.

C.1 No Pre-Mining

In the first scenario, the dishonest party only starts mining the secret branch of the blockchain containing T_d when T_h is submitted to the blockchain. This is the scenario that was analyzed in the Bitcoin whitepaper **[Nak09]**, where an approximate result was obtained, and by Grunspan and Perez-Marco **[GPM17]**, who gave an exact result. The exact result and its proof is repeated here, as it is the basis for the analysis of the remaining scenarios.

Theorem 1 [GPM17]. Let *NoPremine*(*q*,*z*) denote the probability that a dishonest party controlling a fraction $q < 0.5$ of the hashpower succeeds in a double spend attack, assuming the dishonest party starts mining when the transaction T_h is submitted to the blockchain and the honest party waits until T_h is *z* blocks deep before accepting payment, and let $p = 1 - q$.

$$NoPremine(q,z)=1-\sum_{k=0}^{z-1}(p^z q^k - q^z p^k)\binom{k+z-1}{k}.$$

Proof: Let *k* denote the number of blocks mined in the secret branch of the blockchain that includes T_d when T_h becomes *z* blocks deep in the public blockchain. If $k > z$, the double spend attack definitely succeeds. If $k \le z$, the attacker can continue to try to extend the secret branch until it is as long as the public blockchain, so it follows from Lemma 1 that the attack succeeds with probability $(q/p)^{z-k}$ and:

$$\begin{aligned} NoPremine(q,z) &= \sum_{k=z+1}^{\infty} p^z q^k \binom{k+z-1}{k} + \sum_{k=0}^{z} (q/p)^{z-k} p^z q^k \binom{k+z-1}{k} \\ &= 1-\sum_{k=0}^{z}(p^z q^k - q^z p^k)\binom{k+z-1}{k} \\ &= 1-\sum_{k=0}^{z-1}(p^z q^k - q^z p^k)\binom{k+z-1}{k} \end{aligned}$$

which completes the proof. □

[1] Following Nakamoto **[Nak09]**, it is assumed here that the attacker always wins if the secret branch and the public blockchain are of equal length.

C.2 Pre-Mining With An Attack Time Selected By The Honest Party

In the Bitcoin whitepaper, Nakamoto writes **[Nak09]**:

> *The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment.*

While withholding the honest party's public key prevents the dishonest party from mining a secret chain of blocks containing T_h, it does **not** prevent them from mining a secret chain of blocks containing T_d, which is the secret chain of blocks that the dishonest party wants to create. Therefore, the above claim is incorrect and the dishonest party **is** able to pre-mine a secret chain of blocks containing T_d.

The next scenario includes the possibility of such pre-mining and assumes the honest party's public key is given to the dishonest party at a time chosen by the honest party[2].

In order to maximize the length of the secret chain containing T_d relative to the public blockchain, the dishonest party starts mining a secret block containing T_d on top of the current blockchain far in advance of the payment. If the dishonest party succeeds in mining the secret block containing T_d before the next block is added to the public blockchain, the dishonest party attempts to extend their secret chain. As long as the secret chain is at least as long as the public blockchain, the dishonest party keeps building on the secret chain. Whenever the secret chain is shorter than the public blockchain, the dishonest party starts over by trying to mine a secret block containing T_d on top of the current public blockchain. Note that once the block containing T_h is in the public blockchain, the dishonest party must continue building on their secret chain without reverting to the public blockchain.

Lemma 4. For any $i \geq 0$, let $S_i = ((p - q)/p)(q/p)^i$. For any $z \geq 0$, $\sum_{i=z}^{\infty} S_i = (q/p)^z$.

Proof: For any $z \geq 0$:

$$\begin{aligned}
\sum_{i=z}^{\infty} S_i &= \sum_{i=z}^{\infty} ((p-q)/p)(q/p)^i \\
&= ((p-q)/p)(q/p)^z \sum_{i=0}^{\infty} (q/p)^i \\
&= ((p-q)/p)(q/p)^z/(1-(q/p)) \\
&= p((p-q)/p)(q/p)^z/(p-q) \\
&= (q/p)^z
\end{aligned}$$

which completes the proof. □

[2] Following Nakamoto **[Nak09]**, it is assumed here that the time from providing the honest party's public key to the dishonest party until T_h is given to honest miners is negligible relative to the time to mine a new block.

In the following, *payment is made* when the honest party's public key is given to the dishonest party and *payment is accepted* when the honest party provides the goods or services purchased.

Theorem 2. Let *PHT*(q,z) denote the probability that a dishonest party controlling a fraction $q < 0.5$ of the hashpower succeeds in a double spend attack, assuming payment is made at a time chosen by the honest party and is accepted when T_h becomes z blocks deep in the blockchain, and let $p = 1 - q$. For any $z \geq 0$:

$$PHT(q,z)=\sum_{k=0}^{z-1}\binom{k+z-1}{k}[2p^k q^z+(z-k)p^{k-1}q^{z+1}-(z-k)p^{k-2}q^{z+2}].$$

Proof: It will be assumed that payment is made immediately after a new block is added to the blockchain, as this minimizes the probability of a successful double spend attack. For any $i \geq 0$, let P_i denote the probability that the secret chain is i blocks longer than the public chain when payment is made. Because the public blockchain was extended immediately before payment was made, it follows that $P_0 = S_0 + S_1$ and for any $i > 0$, $P_i = S_{i+1}$, where S_i is the steady state probability defined in Lemma 2.

There are two cases:

- Case 1: After payment is made, z blocks are added to the public chain before z blocks are added to the secret chain. In this case, let $0 \leq k < z$ denote the number of blocks added to the secret chain from when payment is made until z blocks are added to the public chain. Also, let $j \geq 0$ denote how many blocks longer the secret chain is than the public chain when payment is made. Note that the double spend attack definitely succeeds if $j + k \geq z$, and it follows from Lemma 1 that it succeeds with probability $(q/p)^{z-k-j}$ if $j + k < z$.
- Case 2: After payment is made, z blocks are added to the secret chain before z blocks are added to the public chain. In this case, the double spend attack definitely succeeds.

Therefore:

$$\begin{aligned}PHT(q,z)&=\sum_{k=0}^{z-1} p^z q^k\binom{k+z-1}{k}[\sum_{j=0}^{z-k-1} P_j(q/p)^{z-k-j}+\sum_{j=z-k}^{\infty} P_j]+\sum_{i=0}^{z-1} p^i q^z\binom{i+z-1}{i}\\ &=\sum_{k=0}^{z-1} p^z q^k\binom{k+z-1}{k}[S_0(q/p)^{z-k}+\sum_{j=0}^{z-k-1} S_{j+1}(q/p)^{z-k-j}+\sum_{j=z-k+1}^{\infty} S_j]+\sum_{i=0}^{z-1} p^i q^z\binom{i+z-1}{i}\end{aligned}$$

and it follows from Lemma 4 that:

$$PHT(q,z)=\sum_{k=0}^{z-1} p^z q^k\binom{k+z-1}{k}[S_0(q/p)^{z-k}+\sum_{j=0}^{z-k-1} S_{j+1}(q/p)^{z-k-j}+(q/p)^{z-k+1}]+\sum_{i=0}^{z-1} p^i q^z\binom{i+z-1}{i}.$$

Also, note that for any $i \geq 0$, $S_i = ((p - q)/p)(q/p)^i = (1 - (q/p))(q/p)^i = (q/p)^i - (q/p)^{i+1}$, so:

$$PHT(q,z)=\sum_{k=0}^{z-1} p^z q^k\binom{k+z-1}{k}[(q/p)^{z-k}+\sum_{j=0}^{z-k-1}((q/p)^{z-k+1}-(q/p)^{z-k+2})]+\sum_{i=0}^{z-1} p^i q^z\binom{i+z-1}{i}$$

$$=\sum_{k=0}^{z-1} p^z q^k \binom{k+z-1}{k}[(q/p)^{z-k}+(z-k)((q/p)^{z-k+1}-(z-k)(q/p)^{z-k+2})]+\sum_{i=0}^{z-1} p^i q^z \binom{i+z-1}{i}$$

$$=\sum_{k=0}^{z-1} \binom{k+z-1}{k}[p^k q^z+(z-k)p^{k-1}q^{z+1}-(z-k)p^{k-2}q^{z+2}]+\sum_{i=0}^{z-1} p^i q^z \binom{i+z-1}{i}$$

$$=\sum_{k=0}^{z-1} \binom{k+z-1}{k}[2p^k q^z+(z-k)p^{k-1}q^{z+1}-(z-k)p^{k-2}q^{z+2}]$$

which completes the proof. □

C.3 Pre-Mining With An Attack Time Selected By The Dishonest Party

Now consider the scenario where the dishonest party is able to pre-mine and to select the time when payment is made. As is shown by the following theorem, in this scenario the dishonest party is **guaranteed** to be able to perform a successful double spend attack if they are willing to wait long enough, but the expected cost of the double spend attack grows exponentially with *z*.

Theorem 3. If a dishonest party controls a fraction $q < 0.5$ of the hashpower, and if payment is made at a time chosen by the dishonest party, the dishonest party is guaranteed to be able to perform a successful double spend attack if they are willing to wait long enough. However, if $EDT(q,z)$ denotes the expected time until the dishonest party pre-mines a secret chain that is *z* blocks longer than the public blockchain, and if $p = 1 - q$, then for any $z \geq 0$: $EDT(q,z)=(1/q)\sum_{i=1}^{z} i(p/q)^{z-i}$.

Proof: It follows from Lemma 2 that for any $z \geq 0$, as the length of the pre-mining attack approaches infinity, the dishonest party will create a secret chain that is at least *z* longer than the public blockchain infinitely often. However, it follows from Lemma 3 that for any $z \geq 0$,

$$EDT(q,z)=(1/q)[z\sum_{i=0}^{z-1}(p/q)^i-\sum_{i=0}^{z-1}i(p/q)^i]$$

$$=(1/q)\sum_{i=0}^{z-1}(z-i)(p/q)^i$$

$$=(1/q)\sum_{i=1}^{z}i(p/q)^{z-i}.$$

which completes the proof. □

Finally, note that for any $z \geq 0$,

$$EDT(q,z+1)/EDT(q,z)=[\sum_{i=1}^{z+1}i(p/q)^{z+1-i}]/[\sum_{i=1}^{z}i(p/q)^{z-i}]$$

$$>[\sum_{i=1}^{z+1} i(p/q)^{z+1-i}]/[\sum_{i=1}^{z+1} i(p/q)^{z-i}]$$

$$=[(p/q)\sum_{i=1}^{z+1} i(p/q)^{z-i}]/[\sum_{i=1}^{z+1} i(p/q)^{z-i}]$$

$$=(p/q)$$

so the expected time required in order to pre-mine a secret chain that is *z* blocks longer than the public blockchain grows exponentially in *z*.

Appendix D: Risk Of FDT Dishonest Miner Attack

A dishonest party can attack an FDT with window size *w*, feerate *f*, and block count *b*, by controlling enough hashpower to create an aligned window that allows the FDT to be satisfied even though the prevailing feerate is higher than *f*. The probability of a successful attack on a specific target window of size *w* is maximized by using the following approach. In the following, the *window starts* when the last block before the target window is added to the public blockchain and the *window stops* when the last block within the target window is added to the public blockchain.

First, the dishonest party starts pre-mining far in advance of the target window by creating a secret chain that branches off the public blockchain. Whenever the length of the secret chain is shorter than the public chain, the dishonest party starts over by branching off a new secret chain on top of the current public blockchain. Then, from when the window starts until the window stops:

- whenever a block is added to the public chain to make the length of the public chain exactly match the length of the secret chain, the dishonest party makes the secret chain public[3],
- whenever the public chain becomes longer than the secret chain and the public chain contains fewer than *b* honestly-mined blocks in the target window, the dishonest party starts over by branching off a new secret chain on top of the current public chain, and
- whenever the public chain is longer than the secret chain and the public chain contains at least *b* honestly-mined blocks in the target window, the dishonest party continues to try to mine blocks on top of the current secret chain.

Then, after the window stops, the dishonest party attempts to extend the secret chain until it becomes as long as the public chain, at which point the dishonest party makes the secret chain public. All blocks in the secret chain that are within the target window must have a median feerate that is no greater than *f*[4].

[3] Again following Nakamoto **[Nak09]**, it is assumed here that the attacker always wins if the secret branch and the public blockchain are of equal length.

[4] The dishonest party can accomplish this by creating artificial transactions with low feerates involving UTXOs that they control.

The proof of Theorem 4 is analogous to, and extends, the proof of Theorem 2. In fact, the probability of a successful FDT attack against a target window of size *w* that must contain fewer than *b* = 1 honest blocks exactly equals the probability of a successful double spend attack against a payment that has depth *w*, when the attack time is selected by the honest party.

Theorem 4. Let *PFDT*(*q*,*b*,*w*) denote the probability that a dishonest party controlling a fraction $q < 0.5$ of the hashpower succeeds in a Feerate-Dependent Timelock attack that creates a specific target window of *w* consecutive blocks that contains fewer than *b* honestly-mined blocks[5], and let $p = 1 - q$. For any $z \geq 0$:

$$PFDT(q,b,w)=\sum_{k=0}^{w-b}\binom{w+k-1}{k}[p^{k+b-1}q^{w-b+1}+(w-b-k+1)p^{k+b-2}q^{w-b+2}-(w-b-k+1)p^{k+b-3}q^{w-b+3}]+$$

$$\sum_{k=0}^{w-1}\binom{w-b+k}{k}p^{k}q^{w-b+1}.$$

Proof: For any $i \geq 0$, let P_i denote the probability that the secret chain is *i* blocks longer than the public chain when the window starts. Because the public blockchain was just extended, it follows that $P_0 = S_0 + S_1$ and for any $i > 0$, $P_i = S_{i+1}$, where S_i is the steady state probability defined in Lemma 2.

There are two cases:

- Case 1: After the window starts, *w* blocks are added to the public chain (and the window stops) before $w - b + 1$ blocks are added to the secret chain. In this case, let $0 \leq k \leq w - b$ denote the number of blocks added to the secret chain from when the window starts until the window stops. Also, let $j \geq 0$ denote how many blocks longer the secret chain is than the public chain when the window starts. Note that the FDT attack definitely succeeds if $j + k \geq w - b + 1$, and it follows from Lemma 1 that it succeeds with probability $(q/p)^{w-b+1-k-j}$ if $j + k < w - b + 1$.
- Case 2: After the window starts, $w - b + 1$ blocks are added to the secret chain before *w* blocks are added to the public chain. In this case, the FDT attack definitely succeeds.

Therefore:

$$PFDT(q,b,w)=\sum_{k=0}^{w-b}p^{w}q^{k}\binom{w+k-1}{k}[\sum_{j=0}^{w-b-k}P_j(q/p)^{w-b-k-j+1}+\sum_{j=w-b-k+1}^{\infty}P_j]+\sum_{k=0}^{w-1}p^{k}q^{w-b+1}\binom{w-b+k}{k}$$

$$=\sum_{k=0}^{w-b}p^{w}q^{k}\binom{w+k-1}{k}[S_0(q/p)^{w-b-k+1}+\sum_{j=0}^{w-b-k}S_{j+1}(q/p)^{w-b-k-j+1}+\sum_{j=w-b-k+2}^{\infty}S_j]+\sum_{k=0}^{w-1}p^{k}q^{w-b+1}\binom{w-b+k}{k}$$

and it follows from Lemma 4 that:

[5] Thus the target window contains fewer than *b* blocks with a median feerate above the FDT's specified feerate *f*.

$$PFDT(q,b,w)=\sum_{k=0}^{w-b} p^w q^k \binom{w+k-1}{k}[S_0(q/p)^{w-b-k+1}+\sum_{j=0}^{w-b-k} S_{j+1}(q/p)^{w-b-k-j+1}+(q/p)^{w-b-k+2}]+$$

$$\sum_{k=0}^{w-1} p^k q^{w-b+1}\binom{w-b+k}{k}.$$

Also, note that for any $i \geq 0$, $S_i = ((p - q)/p)(q/p)^i = (1 - (q/p))(q/p)^i = (q/p)^i - (q/p)^{i+1}$, so:

$$PFDT(q,b,w)=\sum_{k=0}^{w-b} p^w q^k \binom{w+k-1}{k}[(q/p)^{w-b-k+1}+\sum_{j=0}^{w-b-k} S_{j+1}(q/p)^{w-b-k-j+1}]+$$

$$\sum_{k=0}^{w-1} p^k q^{w-b+1}\binom{w-b+k}{k}$$

$$=\sum_{k=0}^{w-b} p^w q^k \binom{w+k-1}{k}[(q/p)^{w-b-k+1}+\sum_{j=0}^{w-b-k} ((q/p)^{w-b-k+2}-(q/p)^{w-b-k+3})]+$$

$$\sum_{k=0}^{w-1} p^k q^{w-b+1}\binom{w-b+k}{k}$$

$$=\sum_{k=0}^{w-b} p^w q^k \binom{w+k-1}{k}[(q/p)^{w-b-k+1}+(w-b-k+1)(q/p)^{w-b-k+2}-(w-b-k+1)(q/p)^{w-b-k+3}]+$$

$$\sum_{k=0}^{w-1} p^k q^{w-b+1}\binom{w-b+k}{k}$$

$$=\sum_{k=0}^{w-b} \binom{w+k-1}{k}[p^{k+b-1} q^{w-b+1}+(w-b-k+1)\,p^{k+b-2} q^{w-b+2}-(w-b-k+1)\,p^{k+b-3} q^{w-b+3}]+$$

$$\sum_{k=0}^{w-1} \binom{w-b+k}{k} p^k q^{w-b+1}$$

which completes the proof. □

Note that Theorem 4 gives the probability of a successful FDT attack against a specific aligned window of *w* consecutive blocks. Obviously, the probability of a successful FDT attack against at least one of *x* consecutive aligned windows is at most *x* times as large as the probability of a successful attack against any one specific window. Therefore, *x* times the probability given in Theorem 4 is an upper bound on the probability of a successful FDT attack on any of *x* consecutive aligned windows.

Some example calculations based on Theorem 4 are given in the following tables and programs that implement the formulas from Theorem 4 are available on GitHub **[Law23c]**.

	log$_4$b=0	**log$_4$b=1**	**log$_4$b=2**	**log$_4$b=3**	**log$_4$b=4**	**log$_4$b=5**	**log$_4$b=6**	**log$_4$b=7**
log$_4$w=0	2.099e-01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=1	5.906e-03	4.240e-01	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=2	1.509e-08	1.675e-06	8.373e-01	N/A	N/A	N/A	N/A	N/A
log$_4$w=3	3.896e-30	4.729e-28	4.614e-20	9.990e-01	N/A	N/A	N/A	N/A
log$_4$w=4	1.266e-115	1.571e-113	3.107e-105	1.723e-73	1.000e+00	N/A	N/A	N/A
log$_4$w=5	0.000e+00	0.000e+00	0.000e+00	0.000e+00	2.529e-286	1.000e+00	N/A	N/A
log$_4$w=6	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	1.000e+00	N/A
log$_4$w=7	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	1.000e+00

Table 4. Probability of a successful FDT attack against a target window assuming 0.10 of hashpower is dishonest.

	log$_4$b=0	**log$_4$b=1**	**log$_4$b=2**	**log$_4$b=3**	**log$_4$b=4**	**log$_4$b=5**	**log$_4$b=6**	**log$_4$b=7**
log$_4$w=0	5.556e-01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=1	1.695e-01	8.125e-01	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=2	3.322e-03	2.276e-02	9.941e-01	N/A	N/A	N/A	N/A	N/A
log$_4$w=3	1.827e-09	1.399e-08	2.237e-05	1.000e+00	N/A	N/A	N/A	N/A
log$_4$w=4	9.652e-34	7.633e-33	2.496e-29	1.078e-16	1.000e+00	N/A	N/A	N/A
log$_4$w=5	5.418e-130	4.322e-129	1.674e-125	1.819e-111	4.024e-61	1.000e+00	N/A	N/A
log$_4$w=6	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	6.001e-238	1.000e+00	N/A
log$_4$w=7	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	1.000e+00

Table 5. Probability of a successful FDT attack against a target window assuming 0.25 of hashpower is dishonest.

	log$_4$b=0	**log$_4$b=1**	**log$_4$b=2**	**log$_4$b=3**	**log$_4$b=4**	**log$_4$b=5**	**log$_4$b=6**	**log$_4$b=7**
log$_4$w=0	9.669e-01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=1	9.053e-01	9.945e-01	N/A	N/A	N/A	N/A	N/A	N/A
log$_4$w=2	7.371e-01	8.687e-01	1.000e+00	N/A	N/A	N/A	N/A	N/A
log$_4$w=3	3.764e-01	4.740e-01	8.473e-01	1.000e+00	N/A	N/A	N/A	N/A
log$_4$w=4	3.829e-02	5.060e-02	1.354e-01	8.908e-01	1.000e+00	N/A	N/A	N/A
log$_4$w=5	1.003e-05	1.359e-05	4.400e-05	2.517e-03	9.768e-01	1.000e+00	N/A	N/A
log$_4$w=6	2.075e-19	2.836e-19	9.798e-19	1.175e-16	1.424e-09	9.999e-01	1.000e+00	N/A
log$_4$w=7	2.446e-73	3.352e-73	1.179e-72	1.730e-70	3.958e-62	9.670e-34	1.000e+00	1.000e+00

Table 6. Probability of a successful FDT attack against a target window assuming 0.45 of hashpower is dishonest.