Public Security Disclosure: Permanent Network Fracturing Vulnerability in Ethereum Classic's MESS Consensus Protocol

Max Sanchez

VeriBlock Foundation
VeriBlock.org
v1.0

Abstract

After previously experiencing a 51% attack in January of 2019[1], **Ethereum Classic suffered three additional attacks during August of 2020 resulting in the double-spending of millions of dollars'** worth of Ethereum Classic[2].

In response to these attacks and the ongoing presence of a 51% attack vulnerability, a variety of updates to the Ethereum Classic blockchain protocol were proposed, and while ongoing discussions about other potential solutions occurred, **MESS[3] ("Modified Exponential Subjective Scoring") was adopted as an immediate solution[4]**.

While MESS does increase the cost of long-range 51% attacks, **it does so at the cost of introducing a significant vulnerability that can result in permanent network fracturing attacks which render the blockchain network useless.**

Under market and mining conditions at the time of initial discovery and disclosure in October 2020, this attack could be performed with a very high probability of success with readily available hashrate rental for less than $10,000 USD. At the time of initial publication in early July 2021 it could still be performed with readily available hashrate rental for less than $50,000.

This paper discloses the theory behind this vulnerability and demonstrates through simulation how **MESS could be exploited in the real world** to permanently bifurcate the chain and prevent agreement of chain canonicalization.

Further, we explain optimizations that attackers can use to **increase the chance of success of their attack, target specific nodes on separate sides of the bifurcation, improve the bifurcation ratio, salvage a failed attack without having to restart the attack, and more easily stabilize the network fracture**, making it more cost-effective and simpler to execute than the naïve attack.

Contents

0 Preface

On October 6th, 2020 (five days before the activation of MESS on Ethereum Classic mainnet), the VeriBlock team sent a responsible disclosure to the developers engineering and implementing MESS explaining this vulnerability in detail.

After discussions with the developers, we wrote a simulation[14] allowing the attack to be demonstrated with configurable assumptions (time to process blocks, node signaling latency, mining power distribution, message processing latency, etc.), and demonstrated through simulation that with reasonable real-world assumptions, the attack was viable.

We also provided specific details on further attack optimizations which could be used to significantly increase the probability of successful attack and allow the rapid recovery of failed attacks to be reattempted rather than requiring a restart, which this publication expands on. While we believe the attack to be sufficiently viable without these optimizations, they are helpful for calculating the true cost (including technical complexity) of performing the attack, which at the time of publication we expect to be approximately $50k for initial bifurcation and stabilization (ignoring attacker opportunity cost).

After eight months in which the vulnerability has not been patched combined with a marketcap that at peak exceeded $15B, we believe public disclosure of this vulnerability is best for the long-term viability of ETC.

The VeriBlock team submitted ECIP-1094 on August 31st, 2020, which provides an alternate 51% attack prevention mechanism based on inheriting Bitcoin's Proof-of-Work security.

This vulnerability intentionally omits an important detail about the exploit (which has no effect on demonstrating the viability of the attack) in order to allow ETC developers additional time to verify and respond to the exploit. ETC developers can contact us at hello@veriblock.com for the full disclosure.

1 Introduction

The Modified Exponential Subjective Scoring ("MESS") consensus algorithm was activated on the Ethereum Classic network on October 11th, 2020, as an immediate response to three successful 51%

attacks against the blockchain network in August of 2020 (following a previous successful 51% attack in 2019).

MESS functions by introducing subjectivity to the evaluation of reorganization candidates, on the assumption that long-range reorgs are the result of malicious attacks rather than normal network behavior, and artificially inflating the cumulative Proof-of-Work of the active chain when considering long-reorg candidates.

This subjectivity in reorganization scoring will lead to two nodes on two different tips (with two different cumulative difficulties) with a far-buried ancestor to *disagree* on the canonical chain, which is significantly different from the operation of traditional Proof-of-Work blockchains using traditional Nakamoto consensus like Bitcoin (or Ethereum Classic prior to the adoption of MESS).

The subjectivity in MESS was known and acknowledged at the time of activation (and it was proposed as a "Weakly-Subjective Finality Solution for Ethereum Classic,")[3] but it was believed that such subjectivity would only prevent an attacker's node from being able to convince the "legitimate" network of nodes to perform a deep reorganization.

However, due to block propagation times (not to mention potential Denial-of-Service attack vectors) inherent in all decentralized blockchain networks, a situation can be created in which some legitimate nodes on the network accept the attacker's blockchain and others do not, creating a network fracture where legitimate nodes disagree on the canonical chain.

It should be noted that traditional Nakamoto-style consensus algorithms can also have temporary states of disagreement between nodes, but the nature of Nakamoto consensus necessitates that the next block will build upon one of the two candidate chains and resolve the split (or two blocks are built simultaneously on different chains, and the 3rd block resolves a 2-block contention, etc.). In the case of MESS however, comparison between the two chains also depends on which chain the existing node believed to be canonical prior to receiving the block which builds on either of the two candidate chains. As a result, two peers exchanging full chain information will not necessarily resolve on the same block (even when one chain has a higher cumulative Proof-of-Work) and will stay in permanent disagreement while both chains continue to grow.

Such a network fracture would prevent the blockchain network from being usable because two (or more if the fracturing attack were repeated) alternative versions of the blockchain would exist, preventing global consensus.

While the implementation of MESS does mean that continued mining on one side of the network split in the absence of mining on the other would eventually result in all network nodes agreeing on a single canonical chain (because the subjective scoring multiple of the local canonical chain against a proposed reorganization is limited to 31), a small amount of mining on both sides of the split (which could either be the result of honest miners mining different chains, or the attacker continuing to point a small amount of hashrate when necessary) would perpetually prevent this agreement on the canonical chain.

In this disclosure we demonstrate through simulation that with reasonable real-world assumptions such a network fracturing attack could be performed, and that the cost to do so would be very small relative to the value and trading volume of the network.

2 MESS Implementation Details

MESS introduces subjective scoring to the comparison nodes make between two chains. When a node using MESS becomes aware of two potential alternate chains, it decides which one is canonical by comparing an adjusted version of each chain's cumulative Proof-of-Work since their common ancestor, giving *significant preference* to the chain they currently consider canonical.

2.1 Subjective Gravity Calculation

The original proposal for MESS calculated the gravity multiplier for the local chain's Proof-of-Work using the following formula:

$$gravity = 15 * \sin\left(\frac{reorganizationTime + 12000\pi}{8000}\right) + 15 + 1$$

Where *reorganizationTime* is the difference between the node's current active chain's tip's timestamp and the timestamp of the most recent common ancestor shared between the node's currently active chain and the proposed reorganization chain.

When considering whether to accept the proposed fork, the cumulative Proof-of-Work of the current canonical chain (since the shared ancestor) is multiplied by this gravity value prior to being compared to the cumulative Proof-of-Work (since the shared ancestor) of the proposed reorganization.

2.2 Subjective Scoring Algorithm

When a node implementing MESS compares a proposed reorganization to its local canonical chain, it calculates a gravity multiplier (as explained in 2.1), calculates the cumulative PoW of its canonical chain's tip and the proposed chain's tip since the common ancestor of the proposed reorganization, multiplies the calculated cumulative PoW of its canonical chain by the gravity multiplier, and then compares this adjusted version of its canonical chain's cumulative PoW to the proposed reorganization's non-adjusted cumulative PoW.

Specifically:

1. Let *currentTip* be the block at the tip of the current active chain
2. Let *proposedTip* be the block at the tip of the proposed reorganization chain
3. Let *commonAncestor* be the highest ancestor that both *currentTip* and *proposedTip* have in common
4. Let *reorganizationTime* be the difference between *currentTip.timestamp* and *commonAncestor.timestamp*
5. Let *gravity* be calculated based off of the approximation explained in section 2.1
6. Let *currentForkDifficulty* be the cumulative difficulty of *currentTip* minus the cumulative difficulty of *commonAncestor*
7. Let *proposedForkDifficulty* be the cumulative difficulty of *proposedTip* minus the cumulative difficulty of *commonAncestor*
8. If *proposedForkDifficulty* > *currentForkDifficulty* * *reorganizationPenalty* then accept the reorganization, otherwise reject the reorganization.

3 Network Fracturing Attack Summary

An adversary who wants to exploit the network fracturing attack would need to build an attacking chain with a cumulative PoW (since the common ancestor shared with the legitimate chain) which falls within very specific bounds such that normal nodes on the legitimate chain would accept the reorganization if they were at a particular block height and reject the reorganization if they are higher.

As explained in sections 4.3 and 4.4, a successful attacker can’t predict the ideal chain to create to cause a reorganization, but is rather going to build a long chain and distribute it to well-connected attacker-controlled nodes who will withhold it until they detect that some of their peers are one block behind others and that a subchain of their attacking chain has the correct cumulative difficulty to cause a successful reorganization in the one-block-behind peers, but not of the peers who have already processed the most recent network block.

Afterwards, the attacker points hashrate to the original chain, or releases more blocks on their attacking chain (or point hashrate to the attacking chain, if they do not have any more precomputed blocks) depending on which chain needs to be extended to prevent either chain’s cumulative difficulty from overcoming the other’s chain with the MESS penalty multiplier applied (which would cause the entire network to settle on one of the two chains).

Performing the attack successfully involves creating custom node software that is aware of the state of remote nodes, can perform MESS calculations of what its peers would do given their current known chain and a subchain of the attacking chain being built, and can propagate the (sub)chain being created by the attacker when they detect that some of their peers would accept the attacking chain, and when others would not.

4 Network Fracturing Attack Considerations

Several variables complicate performing the exploit which a successful adversary would need to account for, and other variables can be manipulated to increase the chance and decrease the cost of successful attacks.

4.1 Difficulty Adjustment Algorithm and Block Time Limitations

(Note: this section demonstrates that the difficulty adjustment algorithm and block time limitations are actually not a concern when exploiting the MESS fracturing attack and is not a concern when designing a MESS fracturing attack).

For the adversary to pull off a successful attack they would need to create a fork with a sufficiently higher cumulative difficulty than the legitimate chain and the applicable MESS penalty.

For an attacker to produce a heavier chain, they must produce blocks at a higher difficulty or date timestamps into the future. However, the difficulty at which they produce blocks is based on the difficulty adjustment algorithm, which takes time to adjust to higher hashrates.

In particular, the difficulty of the next block depends entirely on the difficulty of the most recent block, and the timestamp difference between that most recent block and the proposed next block.

Some rules apply to the timestamps embedded in blocks:

1. The timestamp in each successive block must be strictly greater than the timestamp of the previous block
2. Timestamps are expressed in seconds
3. Although not specified in the original Ethereum Yellow Paper, there is a de-facto standard established by both Geth and Parity/OpenEthereum (although OpenEthereum no longer supports Ethereum Classic) that a block with a timestamp more than 15 seconds (ETH) or 30 seconds (ETC) ahead of the local clock is not allowed (although said block can become activated as part of the canonical chain in the future when the difference between that block's timesetamp and the local clock drops below 15 or 30 seconds, respectively)

Because of the 1-second granularity and de-facto future timestamp limitations, an attacker who wishes to propose a higher-difficulty fork to the network must create sufficient blocks that result in the difficulty being properly adjusted to reflect the attacker's hashrate (or more pedantically, the portion of the attacker's hashrate that they want to reveal to the network at the current time).

Specifically (using the post-Byzantium hardfork difficulty calculation algorithm), the difficulty can increase by 0.048828% per block. *Technically, it can increase following the new uncle rule in Byzantium by 0.097656% if an uncle is also included in the parent block, which an attacker could allocate roughly 2x power to execute, but we will ignore this as we soon prove that the difficulty adjustment ramp allowed by the difficulty protocol is more than sufficient to execute a fracturing attack against MESS*.

The important takeaway here is that an attacker can express their full hashrate very quickly to the network despite the difficulty adjustment algorithm because the delay in difficulty adjustment can be overcome with timestamp manipulation, and the attacker does not need to get the difficulty up to that normally expected of their hashrate to express their full hashrate in cumulative PoW because they can squeeze more blocks in than the difficulty adjustment algorithm normally targets.

As a simple example, if the regular network creates a block every 13 seconds, then even if the difficulty adjustment algorithm was very slow, an attacker could express up to 13x the regular network hashrate immediately by creating 13 blocks at approximately the regular network difficulty (it takes time to adjust) by just creating a new block at every possible timestamp (1-second interval) per block the "legitimate" network produces.

So, it is reasonable to assume that an attacker controlling a certain multiple of the existing network hashrate can immediately express that higher hashrate by producing a chain that represents their full hashrate by creating far more blocks per second than the "legitimate" network.

It is however important to note that the speed of blocks (based on the manipulated timestamp regardless of when the attacker actually produced the block) can cause a significant increase in difficulty *which reduces the attacker's granularity in controlling exactly how heavy their proposed chain is,* because an additional block added to the attacking chain is forced to occur at the high difficulty of the attacking chain.

4.2 Failure Types and Recovery Strategies

For the purposes of simplified discussion, we define three types of failures that can result in the attacker unsuccessfully splitting the network:

Type-1:
Description: Attacker fails to produce a large enough chain to overcome the MESS penalty multiplier
Prevention: Attacker should allocate more hashing power than is originally expected and can deprovision mining power to save on costs if their chain has sufficient luck, see section 4.3 for specific recommendations.
Recovery: Attacker allocates an additional quantity of hashrate to catch up with the main chain, rather than restarting. Depending on the specifics of the failure and the incremental cost of the hashrate network, it is generally more economical to recover the attack with additional hashrate, but in some specific instances (such as the remaining hashrate available for rental being much more expensive than the hashrate already provisioned, or the attacker falling extremely far behind while the main network's MESS penalty multiplier continues to accelerate), it may be optimal for the attacker to restart instead.

Type-2:
Description: Attacker produces a large enough chain to overcome the MESS penalty multiplier at block height *n* but the network progresses to *n+1* before the attacker is able to successfully execute the chain split, or the attacker successfully splits the network, but additional blocks on the original chain reorganize the split peers back onto the main chain.
Prevention: Attacker should use reasonable timestamps (rather than simply incrementing by 1) to make reorganizations by split nodes far more difficult because MESS multipliers are based on timestamp differences between the current tip and the timestamp of the last common ancestor block, so split nodes should have as high of a timestamp as possible on their tip block. If the chain

progressed before the attack is possible then the attacker's next block will be capable of splitting the network.
Recovery: When a type-2 failure happens, the attacker can simply re-try the attack at the next block propagation, assuming they have built a chain with a sufficiently high cumulative difficulty (which is almost always the case, unless they experience so many Type-2 failures they can't keep up with the growing MESS penalty, in which case they can allocate more hashrate). This is because the network is still effectively on the main chain, so it is functionally equivalent to the attack having never been executed (ignoring the state change that nodes now know about the "latest" block in the attacker's chain, which actually works in the attacker's advantage and is a recommended pre-attack optimization so that split peers don't have to validate the PoW headers [and other data] in all of the attack blocks, making the attack easier to pull off before the target nodes update to the latest height mined on the main chain).

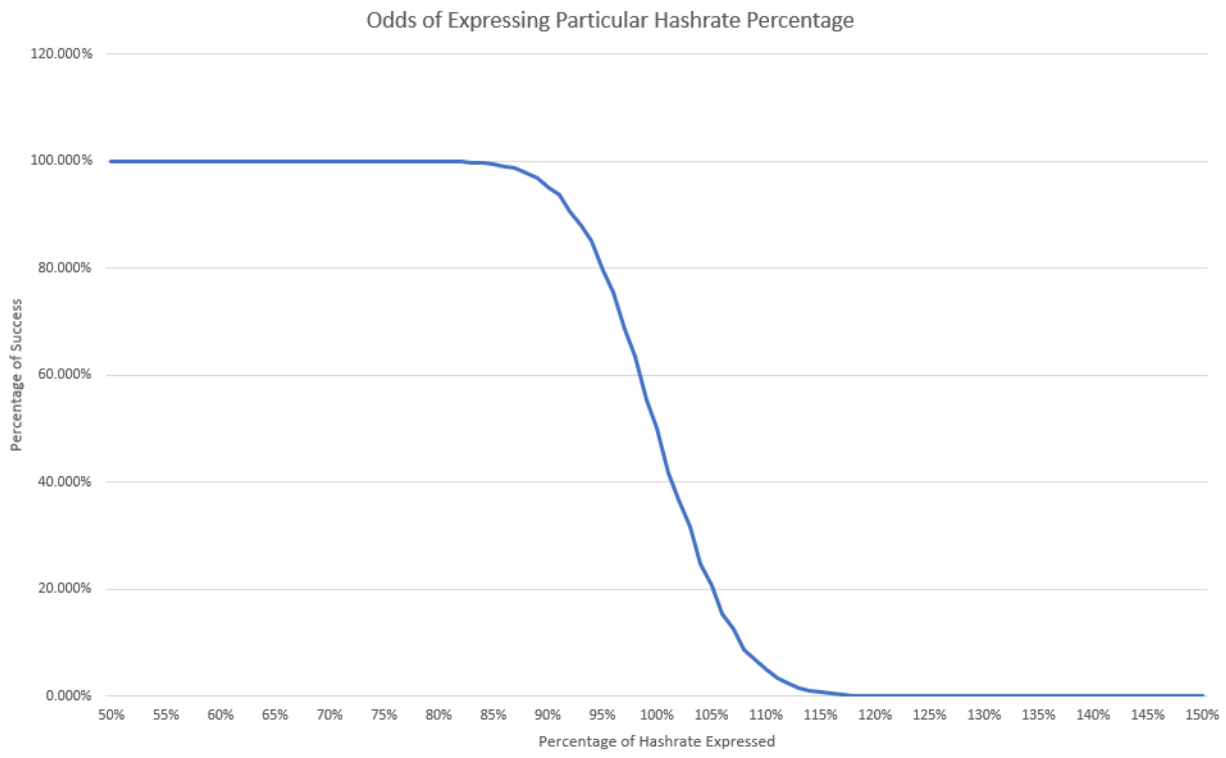
Type-3:
Description: Attacker produces a large enough chain to overcome the MESS penalty multiplier at block height *n*, but successfully reorganizes the entire network onto their attacking chain, so no split occurs (only a deep reorganization). This is equivalent to a standard long-range MESS-adjusted reorganization without intent to cause a chain split.
Prevention: Attacker only propagates their (sub)-chain when rogue nodes detect a portion of the network has already reached the block which would prevent the reorg from succeeding (but some nodes are still on the previous block and would be vulnerable to being split from the "main" network).
Recovery: If attacker's attacking chain has timestamps crafted appropriately, the attacker could switch to mining on the (previous) main chain while the legitimate network mines on the (now canonical) attacker-built chain, and attempt to splinter a fraction of the network from the now-canonical original attacker chain to the abandoned "main chain" which the attacker has switched to be their new attacking chain. Note that this can require more hashing power than required to originally pull off the attack, because the cumulative difficulty of the previously legitimate chain is significantly lower than the cumulative difficulty of the attacker's now-canonical chain. NOTE: a Type-3 attack is less-net-negative (than a traditional absolute failure) for an attacker because they receive all of the mined coins from their attacking chain

4.3 Attacker Mining Variability/Luck

Because Proof-of-Work mining is a process defined by randomness, the amount of cumulative difficulty an attacker produces on their chain will not precisely reflect the amount of hashrate they control.

This issue only exists in one direction: if an attacker has bad mining luck, then they are unable to create a sufficiently high cumulative PoW chain to execute the attack correctly, but if the attacker has good mining luck, they can throw away blocks that cause them to overshoot their attacking chain's weight target.

The odds of an attacker being able to express a certain percentage of their hashrate follows the following distribution:



For example, an attacker with 20 TH of computing power would be able to create a chain that expressed *at least 15 TH* of computing power with 99.99+% success, *at least 16 TH* of computing power with 99.97% success, *at least 18 TH* of computing power with 95.22% success, and *at least 20 TH* of computing power with 50.00% success.

Because a high level of precision is needed in the exact amount of Proof-of-Work generated by the attacker on their chain, *an attacker is likely going to want a roughly 10%+ buffer* (which puts them in the ~95% chance of success range) *of additional hashpower over the hashpower on average needed* to build the attack they want to build to compensate for bad luck. Recall that the miner can always "throw away" (or more pedantically, withhold, which is relevant later) blocks to lower the cumulative difficulty of the chain they build if they have "good" mining luck.

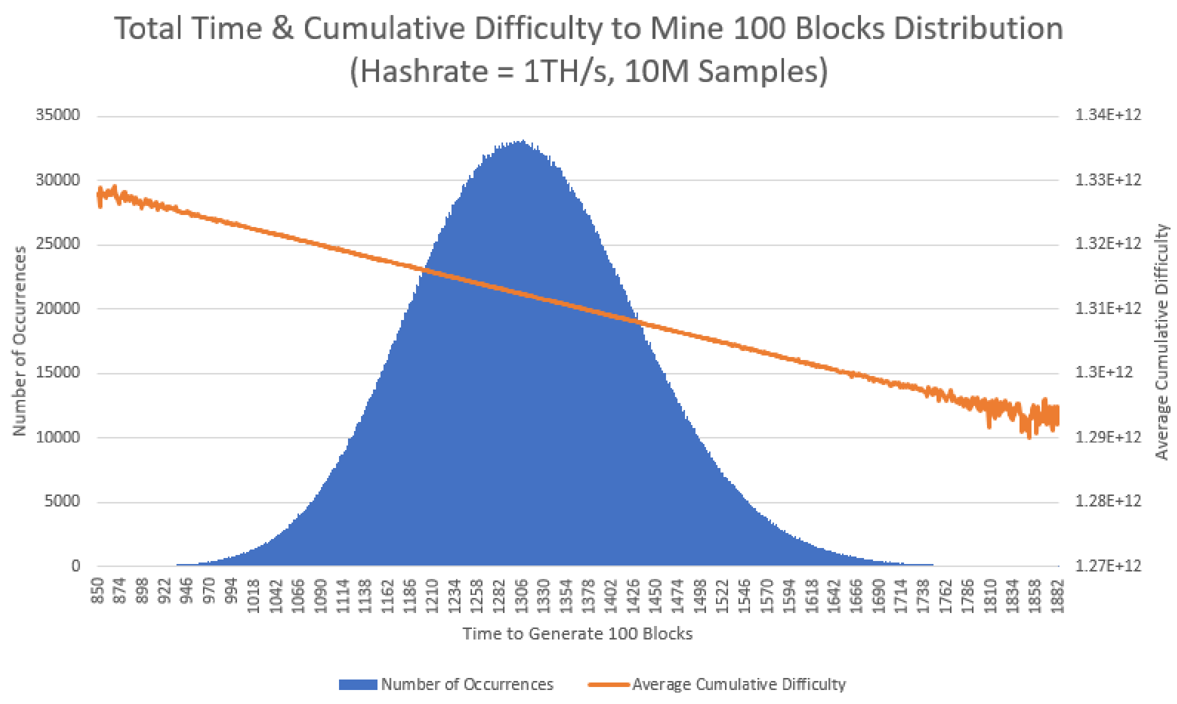
Rather than an attacker building the perfect chain to attack at a particular height, the attacker would need to instead only build a chain which allows them to pull a lower subchain of their attacking fork at the moment when that lower subchain is at an ideal difficulty to fracture the network.

4.4 Legitimate Network Mining Variability/Luck

Fluctuations in network difficulty (due to miners joining/leaving and overall network mining "luck") cause the goalposts of the precise cumulative difficulty of the attacker's chain needed to successfully reorganize some nodes and not others to shift.

As a result, the attacker cannot predict the exact pairing of length and cumulative difficulty of the chain they need to build, because they cannot predict how network mining variability is going to affect the reorganization edge.

Also variability in luck can cause a chain to be produced much faster (higher total cumulative difficulty and lower MESS multiplier) than expected, or much slower (lower total cumulative difficulty and higher MESS multiplier) than expected:



(Note: the "jitter" of the far left and far right average cumulative difficulties is due to insufficient samples [<~200] at the two tales of the distribution due to their extreme rarity).

Instead, the attacker must build a longer chain than the regular network and constantly check (have their sub-agents check) whether a subchain of the fork they built could be used to fracture the network.

Due to the dilation in effective cumulative PoW of the legitimate chain due to MESS adjustment, the gap between two main-chain blocks' (MESS-adjusted) effective cumulative difficulties quickly become larger than the incremental increase in non-MESS-adjusted difficulty of an attacker's chain with only 51% hashrate control, meaning that an attacker's (non-MESS-adjusted) chain will (nearly) always provide a chain-split vulnerability opportunity after a minimal threshold of time since the common ancestor block of the legitimate and attacking network fork of approximately 12 minutes.

For example, if an attacker starts their fork at a difficulty of 100,000,000 and mines every block with a one-second timestamp increase, the attacker's 100th block will have a difficulty increase of 105,002,681 assuming no uncle blocks. However, after 100 blocks (assuming legitimate network hashrate stays somewhat constant), the legitimate network's 100th block will have an effective incremental difficulty increase of ~119,761,144 because the MESS multiplier at 1300 seconds (100 blocks at 13 seconds each, for illustration) is ~1.19761144. It should also be noted that the 100th block of the attacker's chain would not be used to reorganize the chain but rather one slightly lower (because higher blocks in the chain have more cumulative difficulty), which will have an even smaller incremental increase.

As a result, *assuming the attacker has built a sufficiently large chain to have enough cumulative difficulty to reorganize the entire network, then at least one subchain of the attacker's chain will (almost) always have a cumulative difficulty that falls in between the legitimate network's new tip and previous tip.*

4.5 P2P Protocol

In order to successfully pull off the attack, some considerations need to be made regarding the P2P messaging protocol of Ethereum (Classic). In our simulations, we assume RLPx[8] eth/66 but these considerations should otherwise be universal for all post-mainnet versions of RLPx commonly used on mainnet Ethereum forks.

(RLPx version eth/66 only) responses to requests must be sent with the request-id from the requester, so the attacker cannot "pre-send" messages, assuming the request_id is not generated in a predictable order. We assume as per allowances in EIP-2481 that most if not all full-node implementations of eth/66 likely use random 64-bit integers (so they cannot be predicted).

Our simulation assumes the worst-case eth/66 to demonstrate that no protocol version prohibits the attack.

In particular the attacker will have to perform the following procedures[9]:

AdvertiseSubChain (used to keep nodes up to date with the latest non-chain-splitting block of the attacker chain, and to propagate network-splitting block when the opportunity comes):
Slow Version (don't use): Send: NewBlockHashes (0x01), Receive: GetBlockHeaders (0x03), Send: BlockHeaders (0x04), Receive: GetBlockBodies (0x05), Send: BlockBodies (0x06)
Fast Version (use this, attacker can send to all nodes instead of normal $\sqrt{n}$): Send: NewBlock (0x07)

NOTE: The NewBlock P2P command has an included total difficulty as well. For the purposes of this attack we assume lying about this *td* has no benefit to the attacker, although slight differences in the actual implementations of the P2P protocol on different full node software could potentially provide some small benefit to the attacker (and if there were any benefit here it could be easily patched).

LearnAboutPeerHeight (used to determine which height a peer is believed to be at):
Receive: NewBlock (0x07)
or
Send: GetBlockHeaders (0x03), Receive: BlockHeaders (0x04)
NOTE: The second version allows the attacker to actively request remote node state, whereas the first allows the attacker to passively learn about tip state of peers based on what block they advertise to the network as new. Also, the second requires that the attacker knows the *td* of at least the previous block to the highest block in the headers returned to know the true cumulative difficulty the peer is at, while the first includes the td directly in the message. The existing core-geth implementation already keeps a list of peers and their known heights, but the active ("send") version could be used by an attacker to gather tip info from remote nodes who may have received the latest block but not propagated the block to an attacker node because (honest) nodes will only send NewBlock announcements to $\sqrt{n}$ of their downstream peers, and will (after block validation, which

could be delayed if attacker is building DoS blocks on the legitimate chain) send a NewBlockHashes message to the rest of their downstream nodes.

4.6 Attack Stabilization Timestamp Manipulation

As demonstrated in section 4.1, an attacker can express any (reasonable, far exceeding the uses required for this attack) multiple of their hashrate immediately by mining blocks with rapid timestamps.

However, once peers reorganize onto the attacker's chain, it is optimal for them to also apply a high MESS multiple against alternate chains to prevent the attack from collapsing. This can be done by aligning the timestamps of the attacker's generated blocks close to the timestamps of blocks the legitimate network will produce, which would result in split nodes applying a MESS multiplier >1 to reorganizations back onto the "main" chain. This helps to mitigate Type-2 failures and allow higher chances of recovery from Type-3 failures.

We will assume the true mining power of the network remains relatively stable during the attack, but during times of higher hashrate volatility an attacker may prefer to choose a more conservative timestamp manipulation scheme.

Due to the de-facto 15-second future-dating limitation in Ethereum Geth (and likely similar limitations in other implementations) and 30-second future-dating limitation in Ethereum Classic Geth, an attacker should set timestamps conservatively so that the attacker can use sub-chains of their attacking chain to appropriately attack the network without having the subchain's tip timestamp be too far in the future for nodes to accept it.

For example, if an attacker generated a subchain that had a block with cumulative difficulty (since forking point) of 1M and a timestamp of 1000, but the "honest" network transitions over the 1M (MESS-adjusted) cumulative difficulty since the forking point but the network was more than 15 seconds behind the attacker chain (ex: network time 980), then the block which causes the >1M MESS-adjusted total difficulty on the honest chain would have approximately 5 seconds to propagate to the entire network before any honest node (with a synchronized clock) would even consider the attacker's chain as canonical.

From a practical standpoint, a rule to set the timestamps somewhere around 10% lower than when the attacker projects the network will reach a MESS-adjusted difficulty would likely be sufficient without sacrificing too much of the MESS penalty peers who reorganize onto the attacking chain will compare the "honest" chain with.

A more sophisticated attacker could dynamically adjust their timestamps based on the honest network's progression, in case the honest network experiences much better or worse mining luck than expected. This would allow the attacker to recalibrate timestamp calculations for new blocks, allowing the honest network to "catch up" if it experienced bad luck, or allowing the attacker to take advantage of higher MESS multipliers for future attack attempts which would improve post-split attack stabilization if the legitimate network block progression were lagging.

4.7 Block Validation Time and Preemptive Partial Chain Disclosure

The time taken between revealing an attacking chain to a node the attacker expects to split off from the main chain and having the node accept the attacker's fork are critical to ensure the target node accepts the reorganization before it accepts the newly mined block from the legitimate network.

This has three implications:

1. The attacker's nodes should send the highest subchain that will not reorganize any of its peers whenever possible, so that nodes have fewer blocks to receive and process when considering switching to the attacker's chain, increasing the probability of successful attack.
2. The attacker should produce completely empty blocks, so they process quickly on target nodes (this results in nodes that received a NewBlock message quickly sending a NewBlockHashes message to their $1 - \sqrt{n}$ peers they did not immediately propagate the NewBlock message to prior to validation, which speeds up propagation of the attacker's chain).
3. The attacker should flood the ETC network with transactions, so that blocks propagated by legitimate miners will (generally, miners can always elect to mine blocks with empty transactions and sometimes do so when unsure what transactions will be valid in the block they are building while they confirm the new network block) take longer for the network to validate than attacker blocks. A sophisticated attacker can create transactions that result in the highest amount of computational overhead possible within the gas limit.

For the purposes of our simulation, we assume attacker blocks are always empty (worst case for attacker) and that normal network blocks take the same time to process (also worst case for attacker), so without assuming the presence of a denial-of-service vulnerability[6] in the pricing of expensive opcodes[7] in the EVM like SLOAD, BALANCE, EXTCODEHASH, etc. It should be noted that depending on P2P implementation it may be possible to DoS specific (well-connected) nodes' mempools with hard-to-process transactions to delay them from hearing about blocks (as strategic control-points to manipulate propagation of the legitimate chain block that creates the attack opportunity), but we also ignore this vulnerability in this analysis because it only works on implementations which process *all* network messages in order of receipt from any peer, and could be easily mitigated by processing block-related (NewBlockHashes, GetBlockHeaders, BlockHeaders, GetBlockBodies, BlockBodies) queries at a higher priority.

As well, there is a de-facto limit in the current popular full-node implementations for Ethereum Classic and Ethereum that no block which is more than 15 seconds (Ethereum) and 30 seconds (Ethereum Classic) ahead of the current system clock will be accepted (but can later be accepted once they fall within 15 or 30 seconds of the system clock [respectively], which is irrelevant to exploiting MESS because we need our blocks to be accepted nearly immediately), so the attacker cannot reasonably use forward-timestamping techniques to manipulate difficulty or artificially inflate the MESS penalty multiplier nodes who accept their fork will apply to alternate forks relative to wall time.

4.8 Non-Tx DoS Vulnerabilities

There are a number of DoS vulnerabilities an attacker could utilize to increase their advantage, including:

- DDoS-ing pools, bringing their connected hashrate temporarily offline
- DDoS-ing nodes to delay correct acceptance and further propagation of the "honest" chain

4.9 Effects of Uncle Blocks on Difficulty Calculation

(Note, as explained below this section proves that uncle block effects on difficulty calculation have very little bearing on the success of an attack, this section is for informational purposes only).

After the Byzantium hard-fork, a new rule was introduced into difficulty calculation which considered whether the parent block referenced at least one uncle, which would increase the difficulty as if the timestamp difference gap was 9 seconds less than it actually was (numbers are rounded, the actual difficulty algorithm does not use floating point numbers):

	Difficulty Change	
Gap (seconds)	**No Uncles**	**Uncles**
1	0.048828%	0.097656%
	…	
8	0.048828%	0.097656%
9	0.000000%	0.048828%
	…	
17	0.000000%	0.048828%
18	-0.048828%	0.000000%
	…	
26	-0.048828%	0.000000%
27	-0.097656%	-0.048828%
	…	
35	-0.097656%	-0.048828%
36	-0.048828%	-0.097656%
	…	

As long as the attacker's modified nodes know the cumulative difficulty of its peers, it doesn't matter whether the next block progresses the difficulty forward the equivalent of two blocks. It should also be noted that the attacker does not have to wait for the next block to come through to determine whether it built on a parent with uncles, because the parent (which references) the uncles is already available to the attacker.

In fact, the uncles contributing to difficulty adjustment actually provide the attacker with a small benefit: if the next "honest" network block has less than a 9-second gap from the block it builds on, it will increase the cumulative difficulty by 2x what a normal block could have. This gives the attacker a stabilization benefit because the nodes which have not received the current block can be reorganized with the attacker chain ending in a tip that is two (standard) block difficulty adjustments ahead.

4.10 Attacker Node Network Optimization

Attackers should ensure high connectivity between their own “malicious” nodes and other nodes on the honest network. This allows them to detect new blocks (to check for attack viability) as early as possible, as well as reveal the attacking chain as quickly as possible to the vulnerable nodes they intend to split off of the network.

Additionally, all of the attacker’s nodes should be directly connected to each other, rather than all connecting to only a central coordinator server, and attack coordination should be done in a P2P manner where one attacker node sees a network split opportunity and sends the message out to all of the other attacking nodes, rather than relaying information to a central coordination server which then instructs nodes to begin propagating an attack subchain.

4.11 Attacker Information Asymmetry when Mining on an “Honest” Pool

A sophisticated attacker could use a portion of their hashrate to mine on an honest pool on the legitimate network and schedule their attack when the attacker themselves successfully mine a block on the legitimate network that creates the vulnerability opportunity.

The attacker can withhold the winning share from the pool for a short period of time while they instruct their attacking nodes to release the attack, allowing them to control not only the propagation of their attacking chain, but the beginning of the propagation of the new block on the legitimate network which creates the attack opportunity.

Generally, an attacker would initially allocate their full hashrate to building their attacking chain, and once they have a sufficiently long chain to make several fracturing attempts redirect a portion of their hashing power to a pool where they intend to mine one of the blocks that creates the attack opportunity window.

It should also be noted that an attacker may choose the initial propagation point of their block strategically rather than relying on pool connectivity depending on network topology (see 4.14).

4.12 Legitimate Network Topology and Block Propagation

As the attack is based on splitting nodes during the propagation of a new block, the rate at which blocks propagate across the network is relevant.

Blocks cannot propagate on the network faster than messages can be delivered across the network.

For example, if a block originates at peer A with neighbors {B, C} and node D is only peered with neighbors {B, C}, then for the block produced by A to reach D, it can be assumed to take the quickest path (A->B->D or A->C->D) but will always require two hops. Based on the small-world network[13] assumption, a P2P network defined by a graph over set A can be expected to roughly scale mean geodesic distance (“distance” defined as number of hops) by $\log|A|$ (where || signifies cardinality), but it should be noted that high-economic-value nodes [exchanges, explorers, bootstrap nodes, etc.] generally have a far lower geodesic distance between themselves due to incentive for high connectivity (similar to how the “world leader” subset of the global population will have a lower mean geodesic communication distance compared to random population sampling).

For the sake of message propagation analysis in a flood-fill gossip network, a graph of the P2P network set can be transformed to a directed acyclic graph with edge weights determined by messaging latency (if messages are optimistically propagated prior to verification), or edge weights determined by messaging latency + processing time (if messages are pessimistically propagated post-verification).

It should be noted that ETH/ETC only propagate blocks to “downstream” peers (those without the block in question) but this has no effect on the attack (in our simulation, “upstream” peers just ignore block messages for blocks they already have, and during an attack the attacker does not care about sending their attacking block to someone who has already seen it from another source as it has no effect on chain canonicalization).

For the purposes of our simulation, we assume all messages follow the geodesic path (optimal for preventing this type of latency-based attack, so worst case for an attacker) between the originating node and all other nodes on the network (which a pure gossip protocol achieves), even when peer latencies are changing (due to network congestion, etc.).

Intentionally delayed propagation protocols (which can enhance some types of message origin privacy) like diffusion[10] (used by Bitcoin Core) slightly increase the ease with which an attacker can execute a MESS fracturing attack, because they prevent messages from always taking a geodesic path (some nodes may take longer to hear about a block than simply the “shortest” path [“shortest” defined according to network latency and (in the case of pessimistic dissemination) message validation times]).

Ethereum (Classic) nodes do not actually propagate a new block to all “downstream” peers, but only to $\sqrt{n}$ peers *prior to block validation* (which is a combination of optimistic and pessimistic message routing; dissemination to $\sqrt{n}$ and $1 - \sqrt{n}$ peers respectively). Pedantically, optimistic propagation is actually $\lfloor\sqrt{n}\rfloor$ but *floor* has an inverse relevancy versus increasing network connectivity so this can reasonably be ignored when analyzing highly connected networks. Post-validation, standard nodes then send only the block hash to the remainder of peers after successful validation (who then have to request a header [from hash], and then the block body [from header], for a total of 5 P2P messages instead of 1).

This communication slowdown means increased legitimate block propagation time relative to optimal *n* peer propagation from an attacker who can (by virtue of the P2P messages they send) ignore $1 - \sqrt{n}$ propagation message routing to all legitimate network nodes their attacking nodes are directly peered with. As explained prior, our simulation assumes that the legitimate network *also* opts for geodesic n initial propagation instead of $1 - \sqrt{n}$ to demonstrate that updating the messaging protocol to pure flood-fill/gossip does not prevent the attack.

4.13 Geographic Information Availability Asymmetry

As latency between nodes is *loosely* based on geographic location (ex: it generally takes longer for a message to travel from North America to Australia than between two North American nodes), any network split is likely to favor separate geographic locations for either side of the network split. However, latency also applies to messaging between attacker nodes, so an attacker node in NA (for example) would generally take longer to learn about the attack availability than an attacker node in AUS assuming the block originated from an AUS node.

An attacker wishing to specifically split geographic regions from each other can use a block withholding technique where they generate a block on the “legitimate” canonical chain and withhold it. The attacker then informs their nodes in one region to begin propagating the attacking subchain while propagating the new block on the legitimate chain in another region.

In addition to targeting geographic regions, this can also be used to target specific nodes. For example, the attacker could release the block initially to an exchange’s node they are peered with while they release the attacking chain to the rest of the network. The exchange node and nodes 1-hop away are likely to stay on the “legitimate” chain, and a large portion of the remaining network will split away from the exchange node and all other nodes which accepted the attacker’s withheld block on the “legitimate” chain before hearing about the attacking chain.

4.14 Network Mapping

While all of our simulations assume an attacker has no knowledge of the network topology (other than being able to peer well with the network by spinning up many nodes which provide strong connectivity to the legitimate network), knowledge of the topology of the network would allow an attacker to more easily fracture the network.

For example, an attacker could run observer nodes in preparation for their attack to detect the average latencies of blocks mined on popular pools to traverse to different portions of the network, and could construct a reasonable map of network topology, either in anticipation of which nodes they want to perform DoS attacks on to slow legitimate network block propagation, or to determine which sections of the network can be most easily fractured by legitimate blocks produced by popular pools.

Additionally, an attacker can allocate a small portion of hashrate to mine legitimate-chain blocks which they release to specific sections of the network and measure which block drop-points result in the slowest propagation of a block across the network (or which drop-point results in the slowest propagation of a block to a specific section of the network the attacker intends to split off the legitimate chain).

4.15 Distribution of Legitimate Hashrate

As propagation of a found block distributes (roughly) outwards from the origination source of the block (a pool or a solo miner), knowledge of the hashrate distribution on the network (such as major pools) may allow an attacker to craft a more effective attack. Specifically, the attacker could only allow their attacking nodes to begin an attack on a block produced by a specific pool (identifiable generally by “extra data” embedded in the block, protocols like EthereumStratum/1.0.0 don’t allow miners to control block content because only a *header_hash* is provided[11]) that is known for a particular block propagation profile.

Specifically, as detailed in 4.14 an attacker can run observation nodes in preparation for an attack and gather information regarding block propagation speed to different portions of the network based on origin and target a specific pool (or drop their own legitimate-chain block on a specific section of the network) to more closely target their block propagation goals (ex: slower to overall network, slower to specific portion of network).

5 Network Fracturing Attack Detailed Description

To execute a network fracture with the highest possible success rate, the attacker must carefully build their attacking chain using proper timestamp manipulation with an appropriate quantity of hashrate. Additionally, they must release parts of their attacking chain that will not cause a reorganization to optimize the amount of time it takes for nodes to accept the attacker’s fork (without having to process and validate all blocks since the forking point at the time of attack).

5.1 Mining Timestamp Selection

Before the attacker begins mining any blocks, they must select the timestamp interval to use for their blocks. As explained previously, the miner needs to make sure that when they release a (sub)-chain, its tip is no more than 30 seconds ahead of the existing wall time of (essentially) all nodes on the network. Recall that the attacker must choose a very specific subchain to execute the attack successfully at a particular time (or more pedantically, total cumulative difficulty of the honest network), so the attacker must pre-predict what timestamp should be used in each block so that, when usable, the tip will have a timestamp behind global wall time (+30 sec).

The attacker will have to factor in the MESS penalty multiplier of the legitimate chain too: for example at cumulative_honest_difficulty equivalent to 100 blocks and a 13-second average block time, we would expect the multiplier to be approximately 1.19761, so the attacker would want to ensure that the block in their chain which has a 1.19761x difficulty of the existing network difficulty at 100 blocks worth of honest chain difficulty has a timestamp below 1300 (likely a bit lower to avoid good honest network luck from reaching 100 blocks faster than 1300 seconds in the future).

The probability for no more than x blocks to occur in a period of y seconds given a target block time q is:

$$\sum_{i=0}^{i=x} \frac{\frac{y}{e^q} * \frac{y^i}{q}}{i!}$$

For example, the probability that fewer than 100 blocks (assuming network is averaging 13 seconds) will occur in 1300 seconds is 0.5266, the probability that 90 or fewer blocks will occur in 1300 seconds is 0.1714, etc. So if an attacker were to want an 82.86% assurance that their timestamps would be timely [ignoring the 30-second ETC-specific forward timestamping option]), then they would want to assume that at the 1300-second wall clock mark, the network may have progressed only 90 blocks ahead (and should also factor in the small difficulty drop this would cause as well, which can't be perfectly predicted but can be predicted within a range). So, the attacker would make sure that the block with difficulty capable of splitting 90 blocks worth of honest chain difficulty (including the MESS multiplier calculated at timestamp 1300) has a timestamp that is less than 1300 seconds after the timestamp of the forking point.

To be safe, an attacker can choose a dramatically smaller number for the incremental timestamp increases in their block. However, the attacker must keep in mind that they are sacrificing a higher MESS multiplier on the split nodes post-split until at least one block at the correct timestamp is added to the attacking chain (which upgrades the timestamp to approximately the same time as the honest network unsplit nodes).

For the sake of simplicity, we are going to select a 10-second constant timestamp (which conveniently keeps the difficulty constant, although this is not a requirement for successfully pulling off the attack), but more complex dynamic timestamping could be done in responses to good/bad honest network luck while the attacker builds their chain. Note that this timestamp is used instead of the actual real-world time the attacker generated a particular block, because the attacker can choose the timestamps they plant into attack-chain blocks they mine.

5.2 Hashrate Selection

The attacker will need to choose an (initial) amount of mining power to begin building their alternate chain. This amount of hashrate must account for when they expect to execute the attack (likely with a MESS multiplier <1.2, in our simulations we performed splits with MESS multiplier ≈ 1.085) and enough hashrate to compensate for possible bad luck. As explained in section 4.3, an attacker will generally want a 10% buffer over the expected hashrate. As of early June 2021, the ETC hashrate is approximately 26 TH/s. Assuming the attacker plans to fully execute the attack lower than a MESS multiplier of 1.2 (which is extremely realistic, the attack can be successfully pulled off at alternate subchain lengths below 70 blocks, or an approximate MESS multiplier of 1.0853856), the attacker would want to allocate approximately 26*1.2*1.1=34.32 TH/s of mining power.

Note that a sophisticated attacker can start without the buffer for compensating for luck (10% cheaper, in this case), and can allocate more hashrate later in the event that they experience bad mining luck relative to the normal network.

It should also be noted that renting hashrate from a hashrate marketplace may slightly reduce the legitimate network hashrate (which would effectively appear as 'bad luck' while difficulty adjusts on the legitimate network), so an attacker who knows what percentage of ETC hashrate comes from the hashrate rental site they are using can factor this in and rent slightly less hashpower. For the purposes of demonstrating this attack, we do not make this assumption.

After the hashrate is provisioned, the attacker selects a block height to begin the attack from (the latest block on the legitimate ETC network) and begins privately creating a chain off of that base block with their hashrate, propagating new attacking chain blocks to their rogue nodes, who selectively reveal subchains of the attacking chain to the network which are guaranteed not to cause a chainsplit (just to reduce the number of rogue blocks that nodes have to process when the actual fracturing attack is executed).

5.3 Chain Split Execution

At a difficulty of 26 TH/s, the legitimate network difficulty would be 338T.

In one simulation run (with the above hashrate and difficulty), the legitimate chain had the following blocks:

Block	Timestamp	Difficulty	Cumulative Difficulty from Common Ancestor	MESS Multiplier	Adjusted Total Cumulative Difficulty
96	1266	337.1T	32.71B	1.1874313	38.84B
97	1284	337.0T	33.05B	1.1927875	39.42B
98	1286	337.2T	33.38B	1.1933872	39.84B
99	1294	337.3T	33.72B	1.1957955	40.32B
100	1318	337.2T	34.06B	1.2031092	40.98B

And the attacker's chain (with 34.32 TH/s of mining power) had the following blocks (note the attacker in this case used a constant timestamp of 10, so no difficulty adjustments occurred):

Block	Timestamp	Difficulty	Cumulative Difficulty from Common Ancestor
115	1160	338.0T	38.87B
116	1170	338.0T	39.21B
117	1180	338.0T	39.55B
118	1190	338.0T	39.88B
119	1200	338.0T	40.22B
120	1210	338.0T	40.56B

As can be seen above, attacker blocks 115 and 116 can be used to split the network as block 97 is propagating (as it has a cumulative difficulty above 38.84B but below 39.42B), block 117 can be used to split the network as block 98 is propagating (as it has a cumulative difficulty above 39.42B but below 39.84B), etc.

Once one of the attacker's rogue nodes detects that some of its peers are on height n-1 and others are on n, and that there is a valid subchain of the attacker's generated chain which will reorganize peers on height n-1 as explained in the previous paragraph (MESS multiplier multiplied by total difficulty of n-1 is less than the non-MESS-adjusted subchain's total difficulty in question) but not peers on height n (MESS multiplier multiplied by total difficulty of n is more than the non-MESS-adjusted subchain's total difficulty in question) the rogue node will propagate that subchain (or more pedantically the portion of the subchain that its peers have not yet been informed about; recall that the attacker sends out portions of their attacking chain to the network when they wouldn't cause any split in order to reduce processing overhead when the actual attack is executed) to all of its peers, and inform all other rogue nodes to do the same.

5.4 Post-Split Attack Stabilization

After successfully splitting the network, the attacker must stabilize the attack quickly, otherwise too many blocks built on one of the two chains (the original "legitimate" network chain, and the new "attacking" chain which a portion of the legitimate network nodes have been split onto) will cause all nodes to accept a single chain.

There is a chance that the network will stabilize the attack itself, and the higher the MESS penalty multiplier of the attacker's chain (the higher the attacker's chain, but within the forward-timestamp bounds of the network based on wall time), the more likely the attack is to automatically stabilize (and if it requires the attacker to manually stabilize, the easier manual stabilization will be).

As a concrete example, assume the legitimate chain has a *cumulative difficulty* since the forking point of 10000 (100 blocks at difficulty of 100 each, chain split is done between 99 and 100), and the split occurs at *timestamp = 1200* (so *MESS_2 multiplier ≈ 1.1684338* for block 100, and *MESS_1 multiplier ≈ 1.1648108* assuming block 99 had a timestamp 13 seconds in the past). The attacking (sub)chain would have to have a cumulative difficulty between (9900*MESS_1, 10000*MESS_2) = (11532, 11684). If the attacking (sub)chain has a top block with the timestamp 1100 (recall attacker must generate below expected timestamp), then the attacking (sub)chain would have a MESS multiplier of 1.1415736.

As a result, nodes which stayed on the legitimate chain now have a cumulative difficulty of 10000 (on block 100) and will apply a MESS multiplier of MESS_2=1.1684338, so will compare alternate chains' total difficulty against 11684. Nodes which forked to the attackers' chain now have a (non-MESS-adjusted) cumulative difficulty in the interval (11532, 11684), for example 11680.

If another block is built on the attacking chain before another block is built on the original legitimate chain, then the attacking chain's cumulative difficulty will be 11680+100=11780 (it may actually be a bit higher than 100 in some cases, because the attacker's (sub)chain's tip will have a higher difficulty because the attacker produced blocks significantly faster than the target mean time, but attacker can use timestamp=10 to avoid difficulty adjustments for easier math). As a result, all nodes still on the legitimate network would resolve over to the (extended version of) the attacker's (sub)chain because 11780>11684. Note that it is possible if the attacker's block just meets the reorganization minimum, it may be required for more than one block to be built on the attacker's chain to cause the attack to destabilize (a Type-3 failure), as a >1 block progression of the attacker chain may still not cause a full-network reorganization onto the attacker's chain.

It is much harder for regular network conditions to quickly cause nodes which were split onto the attacker's chain to reorganize onto an extended version of the original legitimate chain, because nodes on the attacking chain have a higher cumulative difficulty. In this example, nodes on the attacking chain have a tip cumulative difficulty since the common ancestor block 11680, and would apply a MESS multiplier of 1.1415736, so they would compare the cumulative difficulty of the legitimate chain (on the next block, ~10100) to 13334 (which would require ~33 blocks created on the original legitimate chain without any being created on the attacker's chain).

To stabilize the attack, the attacker should immediately redirect their hashrate to attempt to mine blocks on the legitimate chain, to increase the legitimate chain's score high enough such that subsequent blocks mined on the attacking chain will fail).

6 Network Fracturing Attack Simulation

In order to properly demonstrate the attack, it is required to set up a simulation that accounts for latency between regular nodes on the network, the attacker and the nodes that they control, and the attacker's nodes with each other when they coordinate an attack, in addition to block processing times.

6.1 Simulation Setup

The simulation[14] was setup as follows:

- 100 legitimate peers, each peered randomly with each other (between 10 and 19 connections each), and with random transmission latencies between 0 and 400ms
- 100 rogue peers, each peered randomly with each other and with random transmission latencies between 0 and 400ms, peered with every other rogue node (for sending messages about coordinating attacks) also with random transmission latencies between 0 and 400ms
- A constant *true* network hashrate which exhibits random mining luck
- Two separate runs: one in which network hashrate (block creation) is distributed randomly on the network, and another where blocks are dropped on the 3 most connected nodes (simulating 3 major pools operating on the network). Orphans are allowed to occur.
- A constant *true* attacker hashrate which exhibits random mining luck
- The difficulty adjustment used by Ethereum Classic (post-Byzantium fork) is implemented for blocks added to either the main chain or the attacking chain
- The rogue miner is operating with 125-150% of the network hashrate (tests were done for 125%, 130%, 135%, 140%, 145%, and 150%)
- The attacker produces their attacking chain with manipulated timestamps (each block is ten seconds after the previous)
- As soon as the attacker finds a new block on their fork they broadcast it to their rogue peers (subject to network latency)
- Communication about a block requires three messages to be sent between peers (note that the single NEW_BLOCK could be forced by the attacker, but we ignore this attack optimization for the sake of demonstrating the attack is not a result of the P2P protocol message dissemination algorithm). Technically in the real-world
    - MAKE_AWARE_OF_BLOCK (informs a peer about a block)
    - REQUEST_BLOCK
    - GIVE_BLOCK

- Attacker's nodes automatically trust block headers sent by other peers (so have immediate knowledge of them)
- Blocks on the regular network are full (an attacker could pay transaction fees to fill up the mempool) and take between 100 and 150 ms to fully process (and simulations were also run which demonstrated high rates of success with a 0-time block processing assumption for the legitimate network).
- Attacker's blocks are empty, and so only take between 10 and 15 ms to fully process
- Nodes process blocks in the order they receive them
- Attacking nodes update their known peer heights whenever their legitimate network peers send them a MAKE_AWARE_OF_BLOCK message
- When one attacking node detects that between 5% and 65% of its peers would be fractured by the attack, it sends the calculated subchain of the attacking chain which will result in the fracture to all peers, and informs all other attacking nodes to send that chain as well
- Attacker nodes communicating with each other or the attack coordinator are subject to the same latency as regular nodes on the network communicating with each other (on average)
- The attacker first attempts to perform the attack when the timestamp of the higher block height of the legitimate network is greater than or equal to 857, which corresponds to a MESS multiplier of ~1.0859860

It should be noted that much more pessimistic simulation variables have been successfully used to produce the attack, including a lower attacker:legitimate peer ratio, transmission latencies in the 25-50ms range, zero-block-processing-time assumptions for legitimate blocks etc.

None of our simulations use any of the following optimizations:

1. attack recovery mechanisms (dynamic additional allocation of hashrate, leveraging the original "legitimate" chain as a new attacking chain)
2. additional data sources for block propagation information
3. EVM-level DoS attacks
4. OSI-level (D)DoS attacks
5. attacker network routing being superior to legitimate network beyond trivial direct connection assumption
6. dynamic chain timestamp manipulation (based on legitimate network mining luck) rather than constant timestamping
7. attacker pre-awareness of attack opportunity by mining attack-enabling blocks on legitimate network
8. embedding attack signals in legitimate blocks for improved attack signaling
9. exploitation of standard non-geodesic legitimate block routing due to $1 - \sqrt{n}$ pessimistic routing with 5-layer block message propagation
10. forcing legitimate block propagation slowdowns with P2P table poisoning

6.2 Simulation Results

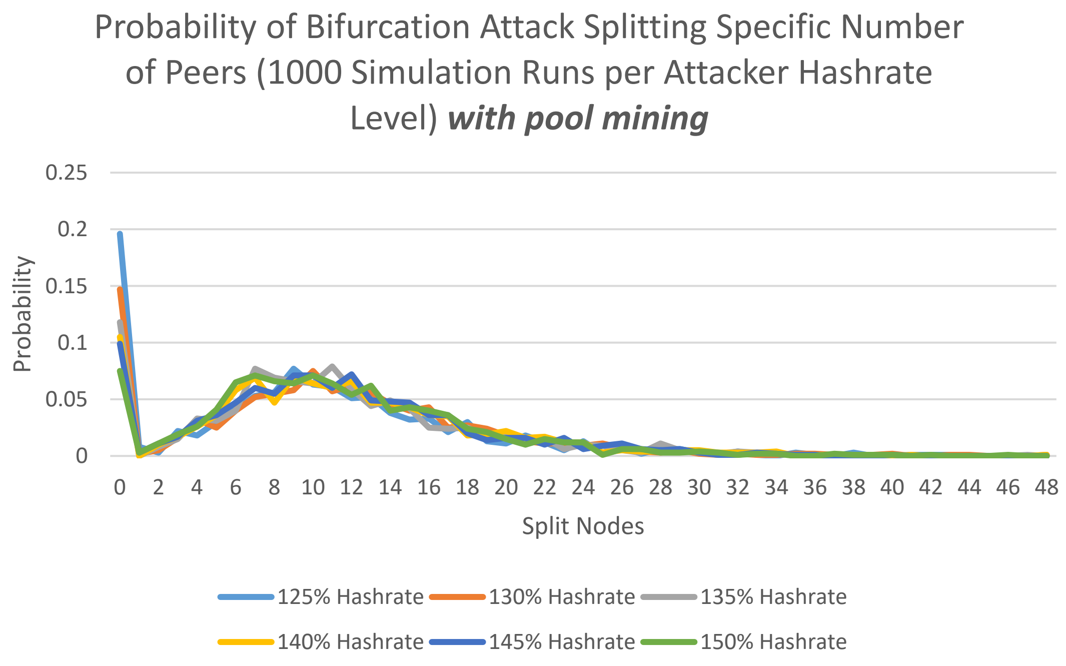
The simulation was successful in fracturing a portion (10-25%) of the network off in most cases, and more importantly demonstrated that after the fracture was created, it could be maintained by the attacker nearly every time. Simulation failures occurred for two reasons: due to a Type-1 failure where the attacker fell behind (bad luck versus legitimate network) which a real attacker could easily mitigate by allocating more hashrate when they fall behind as detailed in 5.2) or due to a Type-3 failure (entire network reorganizes onto attacker fork, but attacker gets back a large portion of their attack budget because the network completely reorganizes to their chain, and they can try the attack again or recover by allocating additional hashrate).

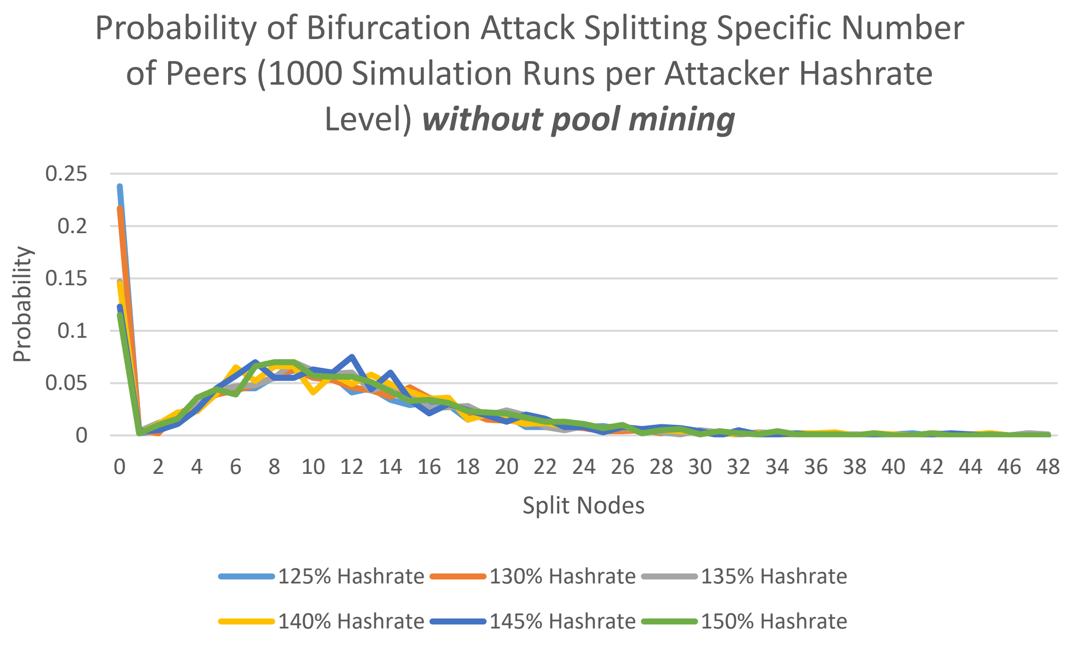
Implementing the other additional attack optimizations described in section 7 would allow the attacker to have an even higher chance of success and fracture a larger portion of the network (closer to a 50/50 split instead of an 80/20-ish split). It should also be noted that the "random connectivity" of the simulation network is a worst-case (lowest probability of attack success), and a more realistic "real-world" P2P graph would be expected to fracture more evenly due to a higher mean geodesic path distance between nodes.

The simulation also demonstrates that reducing the network latency between all nodes (legitimate and attacking) by the same amount has little effect on the potency of the attack; simulations with random latencies of 25-50ms still demonstrated successful attacks.

A successful splitting attack was generally possible in ~15 minutes of simulation time (correlating to the real-world time the attack would take, not the actual runtime of the simulation itself which is much shorter), and another ~2 minutes for stabalization.

Two versions of the simulation were run; one where any node on the legitimate network is randomly selected for the block propagation drop-point (simulating completely decentralized mining), and another where the three most highly connected nodes were the only block propagation drop-points (simulating pool mining behavior). Both simulations demonstrated the network bifurcation vulnerability.





The network split ratio exists independently of the allocated hashrate assuming the absence of a Type-1 failure (which naturally decreases in frequency with increasing hashrate), and with similar results between pool-mining simulation (legitimate block propagation starts at top 3 most connected nodes) and solo-mining simulation (legitimate block propagation starts at random points in the network). Increased hashrate has no correlation with non-failure network split ratios.

Attacker Hashrate = 125%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.196	0.238
5%+	0.753	0.719
10%+	0.496	0.468
15%+	0.231	0.228
20%+	0.102	0.102
25%+	0.043	0.053
30%+	0.019	0.022

Attacker Hashrate = 130%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.147	0.217
5%+	0.797	0.727
10%+	0.567	0.478
15%+	0.271	0.244
20%+	0.112	0.094
25%+	0.050	0.041
30%+	0.020	0.020

Attacker Hashrate = 135%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.118	0.147
5%+	0.826	0.791
10%+	0.542	0.528
15%+	0.247	0.256
20%+	0.114	0.115
25%+	0.052	0.050
30%+	0.018	0.026

Attacker Hashrate = 140%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.105	0.145
5%+	0.840	0.797
10%+	0.560	0.507
15%+	0.276	0.252
20%+	0.125	0.104
25%+	0.046	0.048
30%+	0.024	0.023

Attacker Hashrate = 145%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.099	0.123
5%+	0.839	0.832
10%+	0.570	0.550
15%+	0.270	0.248
20%+	0.116	0.119
25%+	0.052	0.054
30%+	0.014	0.021

Attacker Hashrate = 150%		
Network Split (≥) on Either Side	Probability (Pool Mining Simulation)	Probability (Solo Mining Simulation)
FAIL	0.076	0.115
5%+	0.865	0.821
10%+	0.558	0.531
15%+	0.267	0.269
20%+	0.103	0.126
25%+	0.039	0.051
30%+	0.019	0.021

In summary, with 125% hashrate the attacker **failed** in any network bifurcation with 19.6% probability for legitimate-network pool mining and 23.8% probability for legitimate-network solo mining, and with 150% hashrate the attacker **failed** in any network bifurcation with 7.6% probability for pool mining and 11.5% probability for solo mining.

Otherwise, the attacker with 125% hashrate **succeeded** with a 5%+ network bifurcation with 75.3% probability for legitimate-network pool mining and 71.9% probability for legitimate-network solo mining, **succeeded** with a 10%+ network bifurcation with 49.6% probability for legitimate-network pool mining and 46.8% probability for legitimate-network solo mining, and **succeeded** with a 20%+ network bifurcation with 10.2% probability for legitimate-network pool mining and (also) a 10.2% probability for legitimate-network solo mining.

The attacker with 150% hashrate **succeeded** with a 5%+ network bifurcation with 86.5% probability for legitimate-network pool mining and 82.1% probability for legitimate-network solo mining, **succeeded** with a 10%+ network bifurcation with 55.8% probability for legitimate-network pool mining and 53.1% probability for legitimate-network solo mining, and **succeeded** with a 20%+ network bifurcation with 10.3% probability for legitimate-network pool mining and a 12.6% probability for legitimate-network solo mining.

It should be noted that the results for the 20%+ network bifurcation success with 150% attacker hashrate are anomalous; one should expect a larger sample size of simulations to yield slightly lower probability for a particular threshold of network bifurcation with legitimate-network solo mining versus legitimate-network pool mining. This also demonstrates that attack-producing block drop-points at poorly-connected nodes have a loose correlation with increased attack viability relative to a particular split threshold, and suggests that attack optimizations that target legitimate-network block propagation speeds (7.2, 7.5, 7.6, 7.7) would contribute to increased network bifurcation ratios.

7 Additional Attack Optimizations

A lot of different improvements can be used to give the attacker a higher chance of success, and the ability to cause a larger or more targeted fracture. It should be noted that none of these techniques were implemented in our simulation; our simulation demonstrates the viability of "permanent" network split vulnerabilities assuming none of these optimizations are used.

7.1 Additional Data Sources

By connecting their attacking nodes to pools and explorers to listen to new block activity, an attacker can potentially get knowledge of a block propagating earlier than relying on peers alone.

7.2 Denial-of-Service Attacks

As the attacker approaches a threshold where they are likely to have a valid subchain that can fracture the network between two blocks, they can begin denial-of-service attacks against nodes to slow down block propagation.

7.3 Optimized Routing Between Attacking Nodes

Setting up attacking nodes to have faster communication with each other than just standard peering allows them to communicate when an attack opportunity exists.

7.4 Improved Attacking Chain Timestamp Calculation

By setting the forged timestamps in the attacking chain blocks they produce to values that are closer to what the actual network will be at when performing an attack with that subchain would be viable will let the attacker better protect their fork from being reorganized against prior to them being able to produce one additional block on top, because their attacking fork will have a MESS penalty multiplier greater than 1, so a few rapid original-chain blocks wouldn't cause the fractured nodes to reorganize back onto the main chain with as high of a probability.

7.5 Attacker Mining Block that Produces Attack Opportunity

The biggest hurdle an attacker faces in executing the attack successfully is receiving notice of a new block on the legitimate network quickly and rapidly responding with the correct subchain of their attacking chain (considering subjective locality).

However, the attacker can have first knowledge of the next block on the legitimate network *if they mine it themselves.* Once the attacker builds a long attacking chain (and propagates as much of it to the network as they can without causing a reorganization by staying below the reorganization boundary of all (pedantically 'most' as in <~0.01% chance) nodes on the network), they can use modified mining software to mine at a public pool or use custom pool/daemon software on their own pool that will delay submission of the winning share (or propagation of the winning block) until the attacker's nodes have been given timed instructions to propagate the correct attacker's subchain.

For example, if the legitimate network is at block 999 and the attacker mines legitimate chain block 1000, the attacker can withhold this block for a second while they send their attacking nodes the instruction to propagate the correct attacking subchain tip (assuming all blocks up to that have been already propagated of the attacker's chain to the network to reduce processing time) at a specific time delay. If the attacker is mining to a public pool they would want their attacking nodes to begin propagating the correct attacking subchain tip very shortly after the planned submission of the winning share to the pool. If the attacker is mining to their own pool, they could control block propagation more easily (and target specific known nodes to be on one particular side of the split).

As explained in 4.13 this can be used to target specific nodes for falling on one side of the split, or entire geographical regions (on average).

7.6 Legitimate Block Payloading

The attacker can produce transactions on the legitimate network which carry attack-relevant information. If the attacker is pointing hashrate at an honest pool in an attempt to mine a block which produces an attack vulnerability (or is solo-mining and will drop the found legitimate network block at a strategic point in the network to target specific network splits), the attacker can propagate transactions at high gas fees which carry the attack "signal," allowing attacking nodes that hear about the legitimate network block before hearing about an attack coordination event from other attacking nodes to be informed to start the attack.

Specifically, Ethereum (Classic) transactions can contain a large amount of arbitrary data so the entire attack signal can be easily carried in a single transaction, which could optionally be encrypted with a private key known to the attacker nodes, although obfuscation of the attack signal is pointless if the attacker pre-propagates below-reorganization-depth subchains because that behavior would indicate a bifurcation attack anyway.

It should be noted that (nearly) any blockchain transaction system is capable of carrying an attack signal; even if (theoretically) a blockchain transaction could only send coins to a pre-registered address, an attacker could send coins to specific addresses in a pre-programmed (to attacker nodes) fashion. For example, an attacker could generate and register addresses $\{\{a_{0\_0}, a_{0\_1}\}, \{a_{1\_0}, a_{1\_1}\}, \ldots, \{a_{n\_0}, a_{n\_1}\}\}$, and could include transactions sending to specific addresses to signify specific messages; $\{a_{0\_0}, a_{1\_0}, a_{2\_1}, a_{3\_0}\}$ present in any order in a block could signify the binary message *0010* for example. To be more efficient, the attacker could use existing (generally inactive) addresses and pre-assign them arbitrary values, associate more than two possible addresses in an exclusive data-carrying set ($\{a_{0\_0}, a_{0\_1}, \ldots, a_{0\_n}\}$ can carry n/2 bits), and use parity techniques to maintain message integrity given lack of publication of some transactions or the "accidental" inclusion of some transactions to specific addresses (or "accidental" non-inclusion). This information is provided to demonstrate the applicability of this attack optimization on other blockchains like Bitcoin derivatives were they to use MESS.

If pre-registration (requiring a valid signature) is not required for a given destination address (which applies to nearly all existing blockchain networks including Ethereum [Classic]), then the address itself can be used to encode significantly more information without requiring pre-programming of attacking nodes to associate particular addresses with particular bit patterns. For example, addresses could be fabricated which carry as much data as the non-checksum portion of the address allows. On Ethereum (Classic) specifically, address checksums are implemented as hexadecimal capitalization[12] so the entire address can be used to encode information.

Attackers who are creating their own block (rather than relying on transaction propagation and inclusion) have more flexibility in how they build their block; they have absolute control over "arbitrary data" included in the block (within protocol size-limit restrictions), and absolute control over transaction inclusion which allows them to much more efficiently signal attacks (and control processing time of the block by including specific DoS-causing transactions assuming DoS vulnerabilities in EVM to manipulate block propagation).

However propagated, the attack signal should reference the expected *total_difficulty* range of the containing block or an identifier of the correct previous block hash (so nodes can discard attack messages that are included in blocks they are not meant to be included in), and a possible reference to the *extra_data* section of the block (if a match fails, disregard attack signal) if the attacker wishes to execute an attack only on block propagation from a specific pool which uses a patternistic *extra_data* section.

7.7 Forcing Slow Block Propagation with Peer Table Poisoning

Because Ethereum (Classic) only optimistically sends a new block announcement to $\sqrt{n}$ of their peers and only sends a hash (which requires a total of 5 back-and-forth P2P messages to retrieve the raw block), an attacker who is able to poison the peer table of a node by occupying the $\sqrt{n}$ peer slots chosen for optimistic propagation and can ensure that all downstream peers of the poisoned node will only hear about the block announcement from the poisoned node after the poisoned node fully processes the block, and will require the 5-step P2P block propagation process.

This vulnerability assumes that the subset of the peer table selected is predictable (ex: peer array is populated based on order of connection, so an attacker could poison a node when it starts up), and this vulnerability could largely be mitigated by shuffling the peer table before the selection of the $\sqrt{n}$ subset and randomly electing to propagate optimistically to *n* peers.

It should also be noted that if this vulnerability does exist (assuming Ethereum [Classic] nodes select $\sqrt{n}$ based on a predictable and manipulatable fashion like the list is sorted by initial peering on node startup), it could be used by a malicious miner (regardless of MESS) to de-prioritize the propagation of other miners' blocks, which is particularly damaging given the ratio of the time taken for 5-step P2P block propagation and the ~13-second blocktime target.

8 Attack Cost

The attack we simulated had the attacker commit 125% of the network hashrate for a period of roughly 15 minutes, and then the attacker may have to continue to commit some amount of hashrate to make sure neither fork overtakes the other while waiting for both chains' MESS penalty weights to increase.

However, the amount of hashpower the attacker needs to maintain even in the worst-case rapidly deteriorates to only ~3.226% over the next 6.75 hours (the time it takes for the MESS penalty threshold multiplier to reach 31).

Once 7 hours have elapsed, the attacker only needs to maintain ~3.226% of the network hashrate to maintain the fork indefinitely, although the split of hashrate across the two network partitions is likely to maintain the attack automatically, and as people stop mining the stabilization increases drop dramatically. It should also be noted that because all that matters in preventing the reorganization is the cumulative difficulty of the original and the fork chains, this hashrate can be applied in bursts and "saved up" by strategically allocating hashrate when hashrate rental prices are low.

As of mid-June 2021, the Ethereum Classic hashrate is approximately 26 TH/s, so an attacker would likely want to rent at least 1.25*26=32.5 TH/s of hashing power for 18 minutes. In mid-June 2021, approximately 37 TH/s was available on NiceHash. Assuming a price of 2 BTC/TH/day (which would outbid all current bids as of mid-June 2021 on NiceHash, 1.8 BTC/TH/day would probably be sufficient), 32.5 TH/s for 18 minutes would cost 0.8125 BTC, or approximately $32,500 at a BTC price of $40,000.

Renting the worst-case required 3.226% of the true network (26 TH/s) would be 0.84 TH/s for the 6.75 hour stabilization period would cost an additional 0.4725 BTC, or approximately $18,900 for a total of $51,400. However, it is likely that the partitioned mining power would remove the need for the attacker to maintain significantly less or no additional hashrate after the first 18 minutes.

It should also be noted that a Type-3 failure (where the entire chain reorganizes onto the attacker's chain) will refund the attacker a portion of the cost of the attack, because they receive all the block rewards for their chain (which were 'stolen' from legitimate miners from the reorganization). In current market conditions, the chain the attacker builds in the first 15 minutes of the attack (we can ignore the last 3 minutes because the attacker would not have a network fracture to stabilize) would cost the attacker 0.667 BTC (approximately $26,680) and the attacker would receive approximately 125% of the normal blocks on average (recall attacker can use 10-second timestamp to avoid difficulty adjustments), which would be approximately 86.5 blocks. 86.5 ETC blocks yield a reward of 173 ETC, or approximately $10,000.

References

[1] ETC 51% Attack – What Happened and How it was Stopped (January 2019) [https://bravenewcoin.com/insights/etc-51-attack-what-happened-and-how-it-was-stopped]

[2] Ethereum Classic Hit by Third 51% Attack in a Month [https://www.coindesk.com/ethereum-classic-blockchain-subject-to-yet-another-51-attack]

[3] Agreeing to Disagree: Proposing a Weakly-Subjective Finality Solution for Ethereum Classic [https://medium.com/etc-core/agreeing-to-disagree-proposing-a-weakly-subjective-finality-solution-for-ethereum-classic-7daad47efc0e]

[4] CORRECTING and REPLACING Ethereum Classic Labs Proposes "MESS," an Immediate Network Security Solution for Ethereum Classic [https://au.news.yahoo.com/ethereum-classic-labs-announces-mess-130000199.html]

[5] ECIP 1100: MESS (Modified Exponential Subjective Scoring) [https://ecips.ethereumclassic.org/ECIPs/ecip-1100]

[6] Disclosure: geth/parity DoS transactions [https://hackmd.io/@iwck0wkoSzauVnsYI0h7JA/SkyFmk4_r]

[7] EIP-1884: Repricing for trie-size-dependent opcodes [https://eips.ethereum.org/EIPS/eip-1884]

[8] Ethereum Wire Protocol (ETH) [https://github.com/ethereum/devp2p/blob/master/caps/eth.md]

[9] Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain [https://arxiv.org/pdf/2010.01373.pdf]

[10] Bitcoin's P2P Network [Article] [https://nakamoto.com/bitcoins-p2p-network]

[11] EthereumStratum_NiceHash_v1.0.0.txt [https://github.com/nicehash/Specifications/blob/master/EthereumStratum_NiceHash_v1.0.0.txt]

[12] Ethereum Address Checksum Explained [https://coincodex.com/article/2078/ethereum-address-checksum-explained]

[13] Small-world network [http://www.scholarpedia.org/article/Small-world_network]

[14] VeriBlock Foundation ETC_MESS_Bifurcation_Network_Simulation Github Repo [https://github.com/VeriBlock/ETC_MESS_Bifurcation_Network_Simulation]