TUXEDO: Maximizing Smart Contract computation in PoW Blockchains

SOURAV DAS, University of Illinois at Urbana-Champaign, USA
NITIN AWATHARE, IIT Bombay, India
LING REN, University of Illinois at Urbana-Champaign, USA
VINAY J. RIBEIRO, IIT Bombay, India
UMESH BELLUR, IIT Bombay, India

Proof-of-Work (PoW) based blockchains typically allocate only a tiny fraction (e.g., less than 1% for Ethereum) of the average interarrival time ($\mathbb{I}$) between blocks for validating smart contracts present in transactions. In such systems, block validation and PoW mining are typically performed sequentially, the former by CPUs and the latter by ASICs. A trivial increase in validation time (τ) introduces the popularly known Verifier's Dilemma, and as we demonstrate, causes more forking and hurts fairness. Large τ also reduces the tolerance for safety against a Byzantine adversary. Solutions that offload validation to a set of non-chain nodes (a.k.a. *off-chain* approaches) suffer from trust and performance issues that are non-trivial to resolve.

In this paper, we present TUXEDO, the first on-chain protocol to theoretically scale $\tau/\mathbb{I} \approx 1$ in PoW blockchains. The key innovation in TUXEDO is to perform CPU-based block processing in *parallel* to ASIC mining. We achieve this by allowing miners to delay validation of transactions in a block by up to ζ blocks, where ζ is a system parameter. We perform security analysis of TUXEDO considering all possible adversarial strategies in a synchronous network with maximum end-to-end delay Δ and demonstrate that TUXEDO achieves security equivalent to known results for longest chain PoW Nakamoto consensus. Our prototype implementation of TUXEDO atop Ethereum demonstrates that it can scale τ without suffering the harmful effects of naïve scaling up of $\tau/\mathbb{I}$ in existing blockchains.

CCS Concepts: • **Networks** → **Peer-to-peer networks**; • **Security and privacy** → *Distributed systems security*.

Additional Key Words and Phrases: Proof-of-Work blockchains, Smart Contracts, Scalability

ACM Reference Format:
Sourav Das, Nitin Awathare, Ling Ren, Vinay J. Ribeiro, and Umesh Bellur. 2021. TUXEDO: Maximizing Smart Contract computation in PoW Blockchains. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 41 (December 2021), 30 pages. https://doi.org/10.1145/3491053

1 INTRODUCTION

One major limitation of PoW blockchains such as Bitcoin and Ethereum is that they have useful compute power of orders of magnitude less than a typical computer. For example, our measurements

Authors' addresses: Sourav Das, University of Illinois at Urbana-Champaign, USA, souravd2@illinois.edu; Nitin Awathare, IIT Bombay, India, nitina@cse.iitb.ac.in; Ling Ren, University of Illinois at Urbana-Champaign, USA, renling@illinois.edu; Vinay J. Ribeiro, IIT Bombay, India, vinayr@iitb.ac.in; Umesh Bellur, IIT Bombay, India, umesh@cse.iitb.ac.in.

2476-1249/2021/12-ART41 $15.00
https://doi.org/10.1145/3491053

illustrate (Figure 1a[1]) that the gas limit of each Ethereum block corresponds to a block processing time (τ) of approximately 150 milliseconds. This means only 1% of the block interarrival time ($\mathbb{I}$) of 15 seconds in Ethereum is used for executing transactions. This prevents permissionless blockchains from accepting blocks that contain computationally-heavy transactions.

Computationally-heavy transactions are desirable for many applications. The decentralized and fault-tolerant nature of blockchains has attracted many applications which are moderated using a smart contract. Many of these applications, such as TLS-N [29], ZK-rollups [4], Tornado.Cash [3], require to perform computationally intensive verification. TLS-N requires a smart contract to verify non-interactive proofs about the content of a TLS session. The proof size in TLS-N is currently limited by the restricted gas limit of Ethereum. ZK-rollups seeks to replace the execution of transactions with zero-knowledge proof of its execution, which requires executing computationally intensive transactions.

Problems due to large τ. To see why one cannot arbitrarily increase τ, we must understand the actions taken by a miner upon receiving a new block. The miner first validates the block by executing all its transactions. It then creates its own block whose transactions it executes and finally starts Proof-of-Work (PoW) to mine a new block. In all existing PoW based blockchains, miners needs to perform these steps sequentially. This is because systems like Ethereum require each block to store a cryptographic digest of the latest blockchain state, which miners must verify while validating a block. Additionally, the digest stored in the block also assists users with low computation resources (a.k.a., *light clients*) to efficiently validate, or prove to other light clients, a portion of the latest state. Moreover, it helps new nodes to quickly bootstrap and join the system and makes the system more compatible with stateless cryptocurrencies [10, 33]. Typically CPUs are used to execute transactions, whereas PoW mining uses ASICs [24], and the CPU block processing unit remains idle during ASIC mining.

A consequence of this sequential approach is that higher validation time τ "eats into" PoW time. This gives significant advantages to miners with faster block processing power (i.e. CPU power) than others, and also opens up the system to various attacks. Moreover, as we demonstrate in §7 and Appendix C, with a larger τ, a miner who skips validation of received blocks and/or its own created blocks can mine more than its fair share of blocks relative to its mining power. For example, our experiments show that when $\tau/\mathbb{I} = 0.2$, a miner controlling 33% of the ASIC mining power can mine as much as 68% of blocks on the main chain.

A large τ also leads to the well-known Verifier's Dilemma [23] where a rational miner has to make the hard choice between validating blocks or not. Validating the block reduces the available time it has to mine the next block, and not validating the block comes with the risk of accepting invalid blocks. Moreover, during our evaluation, we observe that (Figure 12) increasing τ leads to a higher backlog of blocks to be processed at miners, which delays block forwarding. This leads to more forks and wasted mining power and lowers the adversarial tolerance of the system [27, 28]. For these reasons, PoW blockchains currently keep block validation time small relative to the block interarrival time, i.e., $\tau/\mathbb{I} \ll 1$. As a result, the CPU block processing unit remains largely underutilized.

Existing works. Existing works that enable blocks with heavy computation do so through *off-chain* solutions [7, 9, 11, 14, 18, 32, 36]. Rather than having all miners execute transactions, which we term *on-chain* computation, these methods delegate computation to a subset of miners or groups of volunteer nodes. These solutions make additional security assumptions, beyond those required for PoW consensus, so that miners can validate the results that volunteer nodes submit. Also, most

[1]Measured by running Ethereum geth client (version 1.9.0) on a virtual machine with 16 cores, 120GB memory, 6.4TB NVMe SSD, and 8.2 Gbps network bandwidth.

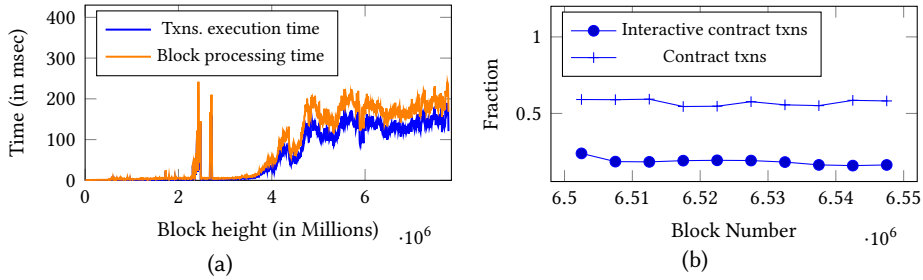


Fig. 1. a) Time taken to validate a received block Ethereum by re-executing the contained transactions. Transaction execution time refers to the time taken to execute only the transactions of a received block. However, block processing time includes both transaction execution time and time taken to write the updated state to the disk. b) Fraction of contract transactions in a block and the fraction of contract transactions which invokes at least one function from a different contract for 50 thousand blocks starting with block height 6.5 Million.

off-chain solutions make restrictive assumptions about the interaction between contracts, e.g., one smart contract does not internally invoke functions of other smart contracts. Such interactions are desirable and often occur in practice. Our measurements of transactions in 50 thousand Ethereum blocks, starting at 6.5 Millionth block, illustrates (Figure 1b) that more than 20% of the transactions addressed to smart contracts interact with other smart contracts. An *on-chain* solution, if designed carefully, can be made to automatically inherit the existing functionality of interaction between smart contracts and also the security guarantees of the underlying blockchain.

Our Approach. In this paper, we propose TUXEDO, the first on-chain solution that can theoretically scale $\tau/\mathbb{I}$ close to 1, i.e., scale computation uses of CPU from 1% to up to 100% while circumventing the problems discussed above. As a result, TUXEDO can increase the useful smart contract computation of PoW blockchains significantly to allow transactions with non-trivial execution time.

The core contribution of TUXEDO is to enable miners to perform most of the block validation and creation in parallel to PoW. We achieve this by allowing miners to delay validation of transactions in a block at height i (B_i) until the arrival of the block at height $i + \zeta$ ($B_{i+\zeta}$) where ζ is a system parameter. Essentially, $B_{i+\zeta}$ contains the cryptographic digest of the state corresponding to the execution of all transactions up to (including) those in B_i. Hence we refer to this approach as *Delayed Execution of Transactions* (DET). This way, the validation of transactions in a block can be done in parallel with the PoW, thereby side-stepping the competition between validation and PoW.

While the idea of DET may seem simple, one faces several challenges when trying to securely adopt it in PoW systems. The first challenge arises due to the variability in the deadline (i.e., the arrival of $B_{i+\zeta}$) for validating transactions (of B_i). As block generation is a random process (often modeled as Poisson), there is the possibility (albeit rare) that an honest miner may fail to execute transactions in B_i before receiving $B_{i+\zeta}$. In that case, the honest miner will not be able to validate $B_{i+\zeta}$ immediately and create $B_{i+\zeta+1}$ when $B_{i+\zeta}$ arrives. As a remedy, in TUXEDO an honest miner always extends its longest *validated* chain.

Second, it is possible that a miner does not have the digest ready for the next block it wants to mine on the longest validated chain. For example, a miner may not have executed transactions included in B_{i+1} and hence does not have the digest ready to put into $B_{i+\zeta+1}$ which it wants to mine. In such situations, TUXEDO allows miners to put a special default state in place of the required state. Unless otherwise stated, we refer to this default state as the *empty state*. We prove in §6.3 that the fraction of blocks with empty state mined by the honest miner can be reduced arbitrarily by setting

ζ appropriately. Using standard techniques from Queuing theory, we prove (§6.2) Tuxedo retains the security (liveness and consistency) of for Nakamoto consensus [12, 16, 19, 27, 28]

Third, an attacker can exploit the fact that a transaction execution state is updated ζ blocks later by broadcasting invalid transactions. Miners may then include blocks containing such transactions in their chains only to find later that they are invalid. Tuxedo addresses this problem by making such an attack expensive. In particular, for every computationally heavy transaction whose execution needs to be delayed, Tuxedo collects a pre-specified amount of fees for the transaction while including it in the blockchain. Later, when the transaction gets executed, miners in Tuxedo consume these fees. Tuxedo confiscates the entire fee of any transaction containing an invalid smart contract and rolls back any state modified by the transaction.

Although Tuxedo parallelizes the execution of computationally heavy transactions and consequently delays their execution, it executes a subset of transactions immediately, i.e., as soon as they are included in the blockchain. We collectively refer to them as the *payment* transactions. Tuxedo requires that the execution time of payment transactions included in every block must be short, i.e., of the order of current block validation time in Ethereum. As in Ethereum, miners then include a subset of them in each block while maintaining that their total execution time is bounded. Users can use this for time-sensitive transactions. Thus we believe that Tuxedo preserves all the original functionalities of Ethereum. We refer reader to §5 for more details on this.

We implement Tuxedo on top of Ethereum (geth client). We perform a comprehensive evaluation of Tuxedo with 50 virtual machines on the Oracle cloud that mimic the 50 top miners of production Ethereum networks that contribute to 99.98% of the total mining power. Our experiments aim to study the improvements Tuxedo provides with increasing τ, i.e., with blocks that require much longer creation and validation time than the existing Ethereum network. Our results illustrates that Tuxedo enables 20× large τ. Furthermore, our evaluation also illustrates that even with such high τ, Tuxedo maintains a mining power utilization of 95.11% which is about 1.5× higher than Ethereum with the same value of τ. Lastly, our experiments also illustrate that, unlike Ethereum, Tuxedo does not benefit miners who skip block validation.

Main Contributions. In summary, we make the following contributions:

- We illustrate through analysis (§3) and experiments (§7) that a naive increase of τ in legacy blockchains gives unfair advantages to miners with faster processing power. An adversary $\mathcal{A}$ can further exacerbate the unfairness by skipping validation of received blocks and creating blocks that it can process quickly.
- We design Tuxedo, a secure on-chain approach that can theoretically scale $\tau/\mathbb{I}$ to 1 in PoW based permissionless blockchains (§4 and §5).
- We theoretically prove security guarantees of Tuxedo under a synchronous network with end-to-end network delay Δ and fixed processing time τ (§6).[2] Our analysis considers all possible strategies by a Byzantine adversary controlling up to $f_{\max} < 0.5$ fraction of the mining power. We also present an approach to choose ζ in order to achieve any desired fraction of honest blocks with a non-empty state.
- We implement Tuxedo on top of the Ethereum Geth client and evaluate it in an Oracle cloud with 50 virtual machines emulating the top 50 Ethereum miners (§7). Our evaluation demonstrates that Tuxedo does not suffer from certain fairness problems, unlike Ethereum, for a high value of $\tau/\mathbb{I}$.

[2]Pass et al. [27] use the term "asynchronous network" for a network with the same constraints.

2 SYSTEM MODEL

We consider a permissionless system consisting of a set of miners. These miners form a connected network and run a blockchain protocol with Proof-of-Work (PoW) as the underlying consensus. All honest miners mine blocks on top of the longest validated chain known to them (see §4.3). Block generation in TUXEDO is assumed to follow a Poisson process with the rate λ where λ depends on the mining power of the network and difficulty of the PoW puzzle. Each miner m_a controls p_a fraction of the mining power. Hence, any arbitrary miner m_a will generate blocks at a rate $\lambda_a = p_a\lambda$. TUXEDO allows execution of Turing Complete programs called *Smart Contracts*. A smart contract can be created by sending a transaction to deploy it on the blockchain. Once a contract appears in the blockchain, its exposed functionality can be invoked by other miners through transactions.

Smart contracts in TUXEDO have unique IDs, and they maintain state, where state corresponds to the unique set of key-value pairs stored at each miner and is controlled by the program logic of the smart contract. In addition to contracts, TUXEDO maintains *accounts*, which maintains tokens. Each account also has a globally unique ID which is the public key of a public-private key pair generated from a secure asymmetric signature scheme. Additionally, TUXEDO has *clients* which own accounts. Clients can generate transactions to create smart contracts, invoke their functions and transfer tokens from one account to another.

Transactions in TUXEDO are ordered in a *Transaction Ordered List* (TOL) and are included in a block. We use T_i to denote the i^{th} TOL. Each miner locally maintains states and updates it by executing a given TOL. Also, we use S_{i-1} to denote the state after executing all the transactions in TOLs $\{T_1, T_2, \ldots, T_{i-1}\}$. With initial state S_{i-1}, we denote the execution of the TOL T_i by:

$$\mathrm{S}_i = \Pi(\mathrm{S}_{i-1}, T_i) \tag{1}$$

where Π denotes the deterministic state transition function that executes transactions in a TOL (T_i in equation (1)) in the order they appear.

Let $\{B_0, B_1, \cdots, B_l\}$ be the blocks known to a miner m_a. In addition to state, m_a maintains a transaction pool $\mathbb{T}_l^{(a)}$, which contains the set of valid transactions created by clients that are yet to be included in a block till B_l. Hereon, when clear from the context, we drop the superscript from $\mathbb{T}_l^{(a)}$ for ease of notation.

Assumptions. We assume the underlying network to be synchronous with end-to-end delay of at most Δ, i.e., all messages sent by an honest miner gets delivered to every other honest miner within time Δ from its release. Also, we assume that all honest miners process blocks in any particular chain serially at the rate of $1/\tau$, where τ is the maximum time needed to validate a block. Like Ethereum, this can be enforced by requiring each transaction to specify the maximum time needed for its execution and keeping a cap τ on the total execution time a block. Note that block processing is different from mining a block; mining involves solving the PoW puzzle, whereas processing is about executing the transactions inside a block. Also, a block with high validation time neither implies a large block size nor that the block has a large number of transactions in it. A small block containing a few computationally-heavy transactions can require a large validation time.

We also assume that the block processing at an honest miner does not contend with PoW, and the honest miners can simultaneously process blocks in distinct forks of the blockchain tree. We envision that our system will be adopted by blockchains such as Bitcoin and Ethereum where block processing can be done using CPUs while mining requires ASICs. Furthermore, the number of simultaneous forks in them are quite small [17].

We assume the presence of an adversary $\mathcal{A}$, who can control up to $f_{\max} < 1/2$ fraction of total mining power of the network and generate blocks at a rate $\beta = f_{\max}\lambda$. Adversarial miners can be Byzantine and can deviate arbitrarily from the specified protocol. The remaining miners are

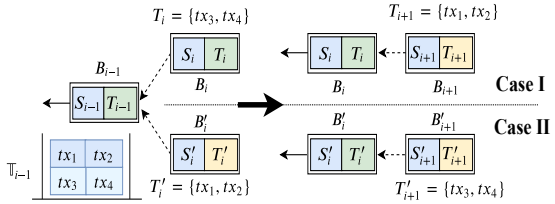


Fig. 2. State transition at an Ethereum miner with $i-1$ blocks on arrival of the i^{th} block. Dashed arrows represent next potential block and solid arrow represents actual arrival of a block. In case I, m_a receives B_i, a block mined by a different miner and in case II, m_a mines the block B'_i by itself.

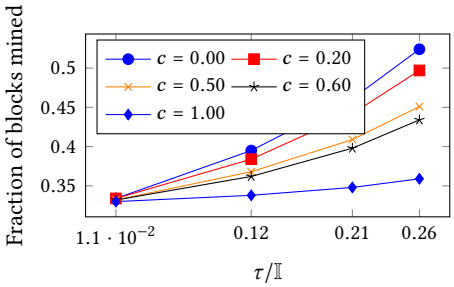


Fig. 3. Fraction of blocks mined by miner m_a with 0.33 fraction of the total mining power, i.e $\lambda_a = \lambda/3$ for varying $c = \tau_a/\tau$.

honest, control the remaining $(1 - f_{\max})$ the fraction of the mining power, generates blocks at a rate $\alpha = (1 - f_{\max})\lambda$, and strictly follow the specified protocol. $\mathcal{A}$ can see every message sent by honest parties immediately and can inject its messages at any point in time. Also, $\mathcal{A}$ can delay messages sent by the honest parties by a maximum of Δ time.

3 BLOCK VALIDATION IN LEGACY BLOCKCHAINS

In this section, we first give some background on the block validation process in Ethereum. Then, we demonstrate why increasing τ leads to reduced fairness in terms of the fraction of blocks mined by an honest miner. This background assists us in identifying the core problem behind smaller block validation time in existing systems. In the later sections (§4) we will describe how Tuxedo securely addresses these issues.

Ethereum is designed to force miners to validate the blocks that they receive. To understand why, we must note that in Ethereum, the state is not stored on the blockchain, and only a cryptographic digest of the state is stored on the blockchain. For a miner to create a potential block of its own, it must include a digest of the new state resulting from its own block in the header. However, the state resulting from executing the transactions of the previous block acts as an essential starting point to obtaining the correct state to put in its own block. Hence the miner is forced to execute transactions in the previous block which it received.

Once a miner successfully creates its own block, it starts mining, i.e., solving PoW on this block. During this PoW two things can happen: *first*, the miner receives a block created by a different miner at the same height as its own potential block; and *second*, the miner successfully solves the PoW, broadcasts its own block with the valid PoW and proceeds to create the next block, extending its very own recent block. We refer to the time spent validating the received block as the *validation* phase; the time spent in creation of next block as the *creation* phase, and time spent in PoW as the *mining* phase. As honest miners do not solve for PoW during the validation and creation phase, we are interested in the time these phases takes to complete. For this purpose, we will first scrutinize these phases more carefully.

If the longest chain known to an honest miner m_a has length $i-1$, with B_{i-1} at the tail of the chain as shown in Figure 2, m_a tries to mine the next block B'_i at height i. Let $\mathbf{S}_{i-1}$ be the state after executing TOLs included till B_{i-1} and $\mathbb{T}_{i-1} = \{tx_1, tx_2, tx_3, tx_4\}$ be the latest transaction pool. For B'_i, miner m_a first selects a TOL T'_i (e.g., $T'_i = \{tx_1, tx_2\}$), executes its transactions in order of their occurrence and starts PoW on block B'_i. Let $\mathbf{S}'_i = \Pi(\mathbf{S}_{i-1}, T'_i)$ be the updated state. Note that till m_a successfully solves the PoW puzzle, the updated state is not committed and remains cached at m_a. As described earlier, while running the PoW for block B'_i, one of two things can happen: either m_a

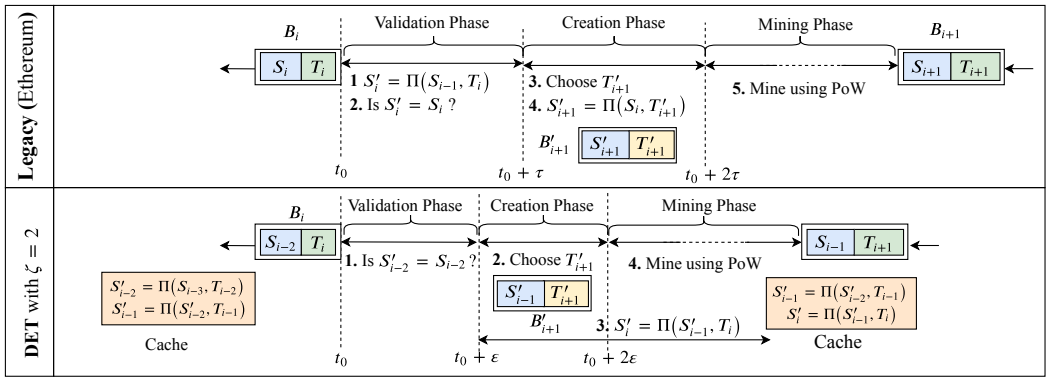


Fig. 4. Actions taken by a miner on receiving a block B_i at time t_0 to validate B_i and create the next block at height $i+1$ in Ethereum (top) and DET with $\zeta = 2$ (bottom). In DET, ϵ denotes the time spent by the miner to validate the received block and create the next one. Since validation/creation in DET involves only a small (constant) number of operations, $\epsilon \ll \tau$.

receives a block B_i at height i mined by a different miner or m_a successfully solves the PoW. We refer to these scenarios as Case I and II, respectively and describe them below.

Case I. Let m_b be the miner of the block B_i (containing ordered list T_i) that m_a receives. Without any coordination between m_a and m_b, it is likely that $T_i \neq T'_i$. In that case, m_a first validates B_i executing all the transactions in T_i. On successful validation, m_a accepts B_i and proceeds to create the next block B_{i+1} at height $i+1$. Also, for block B_{i+1}, m_a picks a new TOL T_{i+1} from $\mathbb{T}_i = \mathbb{T}_{i-1} \setminus T_i$. We illustrate this in Case I of Figure 2.

Case II. Unlike Case I, in Case II, miner m_a commits the state $\mathbf{S}'_i$ and proceeds to create the block B'_{i+1} at height $i+1$ with a new TOL T'_{i+1} from $\mathbb{T}_{i-1} \setminus T'_i$. In our example, $T'_{i+1} = \{tx_3, tx_4\}$. m_a then executes transactions in T'_{i+1} and starts PoW for B'_{i+1}. We illustrate this in Case II of Figure 2.

Another way to describe the block validation and creation mechanism is through the time intervals during which a miner validates the received block, creates the next block and performs PoW. Let t_0 be the time instant of the arrival of the block B_i. Also, let τ be the validation and creation time of a block. In case I, m_a validates B_i during time $(t_0, t_0+\tau)$, creates the next block B'_{i+1} during time $(t_0+\tau, t_0+2\tau)$, and only at time $t+2\tau$ starts PoW for B'_{i+1}. Alternatively, in case II, since m_a himself created the block B'_i, it skips validation of B'_i and spends the time $(t_0, t_0+\tau)$ in creating the next block B'_{i+1}. Additionally, m_a starts PoW for B'_{i+1} at time $t_0+\tau$. The top half of Figure 4 demonstrates the timings along with the computations a m_a performs for Case I in more detail.

3.1 Consequences of high τ in legacy PoW Blockchains

Ideally when all miners are honest and there is no network delay, one would expect that the fraction of blocks mined by a miner should be proportional to its mining power. In this section, we demonstrate that is not the case and show that when $\tau/\mathbb{I}$ is high, the fraction of block mined by honest miners heavily depends on their relative transaction processing speed and their relative mining powers.

Observe from Case II that the creator of a block spends τ units of extra time (i.e., between $(t_0+\tau, t_0+2\tau)$) for PoW while the remaining miners are busy creating the next block. This extra time τ increases its chances of mining the next block as well. This effect gets exacerbated if the miner controls a large mining power (say 30%), because the miner will naturally mine blocks frequently and each of these blocks gives it an advantage to mine the next one as well.

More concretely, let λ_a and τ_a be the block mining rate and block processing time of miner m_a, respectively. Let $c = \tau_a/\tau$ where $0 \leq c \leq 1$, i.e c is the ratio of block processing time of m_a and remaining miners. $c = 0$ implies that m_a can process a block instantly independent of τ. Hence, m_a will spend only $2c\tau$ units of time in case I before starting PoW for the next block. Similarly, in case II, m_a will spend only $c\tau$ units of time before starting PoW. Building on this intuition, we theoretically compute the fraction of blocks m_a will mine in the longest chain for any given choice of λ_a and c. We provide our detailed analysis in Appendix C and describe the main result below.

Figure 3 illustrates results from our theoretical analysis. For example, with $\tau/\mathbb{I} = 0.26$, an miner who controls 30% of the mining power and can validate or create blocks twice as fast as others, i.e. $c = 0.50$, will mine at least 46% of the blocks. Furthermore, a miner who skips both validation and creation of blocks, i.e with effective $c = 0$ will mine at least 53% of the blocks with just 33% of the mining power. We also measure the same using our experimental setup described in §7 and observe that the network delay exacerbates the attack and allows $\mathcal{A}$ to mine 68% of the blocks.

4 OVERVIEW OF TUXEDO AND MAIN IDEAS

In this section, we first give an overview of Tuxedo. Then we describe the concept of *Delayed Execution of Transactions* (DET), the core component of Tuxedo, that enables us to make $\tau/\mathbb{I} \approx 1$. Next, we describe a mechanism to make DET robust against varying block interarrival time and a Byzantine adversary. Finally, we describe the fee collection mechanism of Tuxedo.

4.1 Overview

Recall that a large block creation, block validation time (τ) is unsuitable for a system like Ethereum. In Ethereum, while creating a block, a miner executes all its transactions and puts the cryptographic digest of the resulting state in the block itself. Likewise, miners receiving a block execute all its transactions to verify its digest. Since block validation, block creation, and PoW mining are sequential, a large creation or validation time eats into PoW mining time, opening up the system to unfairness and attacks.

In Tuxedo, instead of making miners execute transactions and reporting a cryptographic digest of the updated state immediately, we delay the reporting of the digest by $\zeta \geq 2$ blocks. In particular the resulting state after executing all transactions up to a block at height i, B_i is included in a block at height $i + \zeta$. Intuitively, this allows miners to process received blocks and create new blocks parallel to the PoW mining phase. Thus larger τ does not eat into PoW time.

A large τ, however, introduces new scenarios not encountered in PoW systems with negligible τ, such as those studied hitherto [19, 27, 28]. In one scenario, an honest miner may not be able to validate the digests present in its longest known chain. This happens if the longest chain has $B_{i+\zeta}$ as its last block and the miner has not yet executed transactions in B_i. This can happen either because subsequent blocks were generated quickly or privately by an adversary and released all at once. As a remedy, in Tuxedo, a miner mines not on the longest known chain, but on the longest chain it has validated so far.

In another scenario, a miner may be able to validate all blocks in a chain and yet not have the state ready to put in the next block it wants to mine. For example, if the last block is $B_{i+\zeta}$ and the miner has executed transactions in B_i but not B_{i+1}, then it can indeed verify the state in $B_{i+\zeta}$ but does not have the state to put in $B_{i+\zeta+1}$. Tuxedo remedies this situation by allowing miners to report an empty state (e.g., all zeros) in the block they create.

We show in our analysis §6, that with the above remedies, the mining of honest miners never stalls, independent of any adversarial strategy. We exploit this fact to lower bound the guarantees of Tuxedo with known guarantees of PoW based Nakamoto consensus [19, 27, 28].

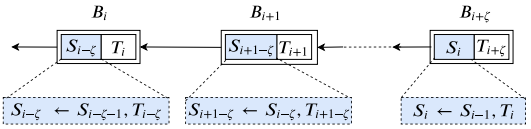


Fig. 5. Structure of blocks with Delayed execution of Transaction with ζ blocks, where the state update corresponding of transaction ordered list of a block B_i gets reported in the block $B_{i+\zeta}$.

4.2 Delayed Execution of Transactions

The main idea behind Delayed Execution of Transactions (DET) is to decouple the inclusion of transactions in the blockchain from the reporting (and hence validation) of the digest of the state resulting from their transactions. In Tuxedo transactions are ordered in a block without being immediately validated, and the digest of the state resulting from their execution is reported ζ blocks later. More formally, a block B_i at height i contains TOL T_i and state $\mathbf{S}_{i-\zeta} = \Pi(\mathbf{S}_{i-\zeta-1}, T_{i-\zeta})$. Hence miners have a window of ζ blocks to compute the state required for validation, and this computation can be done in parallel with the PoW. Figure 4 illustrates this for $\zeta = 2$ where the resulting states are delayed by 2 blocks.

As explained in Case I in Section 3, in existing blockchain designs, up to 2τ time can "eat into" the PoW time. Thus, in order to get $\tau/\mathbb{I} \approx 1$, we need to delay the execution of transactions by at least two blocks. In the rest of this section, we first describe DET with $\zeta = 2$ and then explain why an even larger ζ is needed.

DET with $\zeta = 2$. Let B_{i-1} with state $\mathbf{S}_{i-3}$ and TOL T_{i-1} (as $\zeta = 2$) be the latest valid block known to miner n_a (see Figure 4). For now, assume that n_a has already computed (i) $\mathbf{S}'_{i-2} = \Pi(\mathbf{S}_{i-3}, T_{i-2})$ and (ii) $\mathbf{S}'_{i-1} = \Pi(\mathbf{S}_{i-2}, T_{i-1})$ and cached them prior to the arrival of the block B_i. Here $\mathbf{S}'_{i-2}$ and $\mathbf{S}'_{i-1}$ are the states locally computed by n_a for TOL T_{i-2} and T_{i-1} respectively before arrival of block B_i (ref. Figure 4). Upon arrival of B_i, n_a validates it by checking whether the reported $\mathbf{S}_{i-2}$ matches $\mathbf{S}'_{i-2}$ (step **1**). If it does, then n_a accepts B_i and starts computing $\Pi(\mathbf{S}'_{i-1}, T_i)$ (step **3**). Simultaneously, n_a picks a new TOL $T'_{i+1} \in \mathbb{T}_{i-1} \setminus T_i$, creates the block B'_{i+1} by fetching the precomputed state $\mathbf{S}'_{i-1}$ from its cache (step **2**), and starts PoW for block B'_{i+1} (step **4**). This way, upon arrival of a block, n_a is able to start PoW for the next block immediately.

4.3 Handling variable block interarrival

If blocks arrive exactly $\mathbb{I}$ time apart from each other, then $\zeta = 2$ will be sufficient to scale $\tau/\mathbb{I} \approx 1$. However, in reality, block interarrival times are random and can even be manipulated by the adversary to some extent. In case a sequence of ζ blocks following B_i with TOL T_i arrive closely spaced to each other, it is possible that a miner will not be able to compute the state $\mathbf{S}_i$ before receiving $B_{i+\zeta}$. Hence, the miner will not be able to immediately validate $B_{i+\zeta}$. Without any precautionary measure, in such a situation, miners will be forced to defer creation of the next block, and hence the PoW on it till it computes $\mathbf{S}_i$. If a large fraction of honest miners temporarily stop mining, an adversary $\mathcal{A}$ with faster block processing power will effectively enjoy higher fraction of mining power and may even pull off the "51% attack" during these periods. We address this issue by making two critical observations: *first,* the probability of this event occurring decreases with increasing ζ, and *second*, we can ask honest miners to mine on the longest validated chain during such scenarios. We elaborate on these below.

Need for higher ζ. To see why increasing ζ reduces the probability of the above mentioned undesirable event, we model DET as a queuing system where the transaction processing unit of a miner is analogous to the queue's server. Each arriving block is a task input to a queue and each

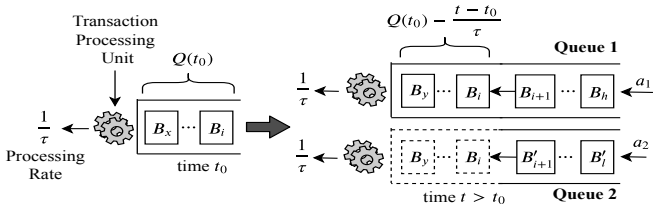


Fig. 6. Queue(s) at a miner at time instant t_0 and $t > t_0$ where the transaction processing unit at the miner process blocks in a queue at a rate $1/\tau$. In case of a fork, as shown on the right part of the Figure, i.e. B'_{i+1} and B_{i+1} forking from B_i, miner maintains an additional queue for the new fork. The dashed part of the Queue 2 i.e. blocks $\{B_y \cdots B_i\}$ need not be processed explicitly and results from processing these blocks from Queue 1 can be directly used. Here a_1 and a_2 are arrival rates to Queue 1 and Queue 2 respectively.

block is processed in τ units of time. In the absence of an adversary and network delays, the block arrival follows a Poisson process with rate α. Assuming the input rate is independent of the queue size, this is essentially an M/D/1 queue. The sequence of blocks that a miner is yet to process in a given chain represents the contents of the queue.

If an arriving block enters a queue of size less than or equal to $\zeta - 2$ then its own state as well as that of the subsequent block have been pre-computed. The probability of the queue exceeding $\zeta - 2$ is the probability of the miner missing the deadline for computing the state which that block must contain. This tail probability of the queue decreases with increasing ζ, thus making larger ζ is more desirable. However, there is a trade-off here, because a larger ζ implies that blocks update the global state later, which is undesirable from a user's point of view. Hence ζ must be chosen to balance this tradeoff. We leave detailed queuing models that take into consideration input variation of blocks due to Δ, the presence of $\mathcal{A}$, and the trade-off due to larger ζ to §6.

Remark. Due to forks, miners in Tuxedo maintain multiple queues, one for each forked chain (see Figure §6), and process them in parallel. Blocks which are common to multiple chains (e.g. blocks $\{B_y, \cdots, B_i\}$ in Fig. 6 need to be processed only once. All our analysis will be valid with multiple queues because, as assumed in 2, a miner processes them in parallel and the input to each queue is still upper bounded by the block generation rate of miners. We believe that the assumption of parallel processing of forks is a reasonable one because, in practice the number of simultaneous forks in them are quite small [17] and is limited by the block-generation capability of the adversary. Lastly, we envision that our system will be adopted by blockchains such as Bitcoin and Ethereum where block processing can be done using CPUs while mining requires ASICs. Thus PoW mining and block processing do not compete for the same resources.

Extending Longest Validated Chain. Although higher ζ lowers the probability of queue of the honest miner crossing $\zeta - 2$ in the absence of an adversary, additional care needs to be taken to provably prevent a Byzantine adversary from sabotaging the protocol. Thus we modify the chain selection rule of Tuxedo from a standard longest chain selection procedure. Recall from 4.1, the following changes are very crucial in lower bounding the chain growth property of Tuxedo which in turn is the core component in the security analysis of any PoW based blockchain including Tuxedo. Refer to §6 for the detailed security analysis of Tuxedo.

Honest miners in Tuxedo extend the *longest chain they can validate*. If a miner does not have the state to put in the next block, it puts a protocol specified default state, such as a sequence of zeros in place of the required state. Unless otherwise stated, we refer to this default state as the *empty state*. We refer to such blocks as *ES* blocks (i.e. blocks with an empty state). Similarly, we refer to blocks with the non-empty state as *non-ES* blocks. ES blocks can contain transactions (see §5) and non-ES block at height i report $S_{i-\zeta}$. Also, during the entire duration, honest miners continue to

process all unprocessed blocks with correct PoW that appear in a chain longer than the current mining head. On successfully validating a block, a miner re-configures its mining head to pick the new longest known validated chain.

4.4 Fees collection in Tuxedo

Since computationally-heavy transactions are executed only after their inclusion in the blockchain, it may be possible for transactions that do not pay sufficient transaction fees to enter the blockchain. We address this by requiring every computationally heavy transaction to specify the maximum computation resources, in terms of gas, needed for its execution. Based on this specification, the fee of every transaction in the i^{th} block, B_i is collected during its inclusion. Clients use the native token of Tuxedo, similar to Ether in Ethereum, to pay for the transactions fees. Once the transaction gets executed, any excess fees, i.e., fees of unused computational resources, are refunded in block $B_{i+\zeta}$ [3] This ensures that only transactions that can pay sufficient amount of gas, a unit of payment in Ethereum, for their transaction fees enter the blockchain.

Note that additional care needs to set the minimum transaction fee paid by a transaction. In particular, we cannot levy a small fixed fee for every transaction as in Ethereum, as such a design can lead to underutilization in terms of the actual amount of gas usage. For example, malicious users can over-specify the required computation resources and later use only a tiny fraction of them. One way to discourage such behavior is to take fees equivalent to the minimum of a α ($0 < \alpha \leq 1$) times the specified gas usage and the actual amount of resources used by the transaction. One may also consider alternative fee mechanisms depending upon the specific use cases. We leave the detailed analysis of the fee collection mechanism as future research.

Similar to Ethereum, Tuxedo also allows its smart contracts to transfer and receive tokens. However, since the transactions of the i^{th} block, T_i, are executed after the fees are collected for $\zeta - 1$ future blocks, additional care needs to be taken to prevent fees of future blocks, i.e., fee transactions in block B_{i+1} to $B_{i+\zeta-1}$, from altering the execution results of T_i. Specifically, Tuxedo restricts its smart contracts from using the native token. However, at the same time, Tuxedo allows its contracts to create their own tokens reminiscent of ERC'20 tokens in Ethereum and use them during execution. These tokens could be contract-specific, shared by several contracts, or shared by all contracts and is up to the contract designer. In our implementation, every smart contract uses the single ERC'20 token: the token2. Also, here on, we will refer to the native currency as the token1. As we mentioned above, separating these two tokens ensures that smart contracts update the state sequentially in the order they appear in the blockchain. We describe the details of our implementation in §5. Note that Tuxedo the two token approach is one of many possible designs Tuxedo adopts to address this issue. A method which uses a single token to address this issue will be described in detail in an extended version of the paper.

5 DETAILED DESIGN

Accounts and Tokens. Our implementation of Tuxedo has two kinds of accounts: user accounts and contract accounts. Each user account maintains both token1, the native currency, and token2, the ERC'20 token to be used in contracts, whereas contract accounts only maintains token2. contract accounts maintain executables that can be invoked by transactions.

Transactions. Each transaction tx of Tuxedo is a tuple containing {type, to, from, gas, $\star$} where type either takes the value 1 or 2, to (resp. from) is the receiver (resp. sender) address, gas specifies the maximum amount of gas tx can use and $\star$ represents the auxiliary information required for

[3]We leave the exact refund policy as a design choice as we primarily focus on the capability of refunding fees if needed. Tuxedo will work same for schemes that does not refund fees as well.

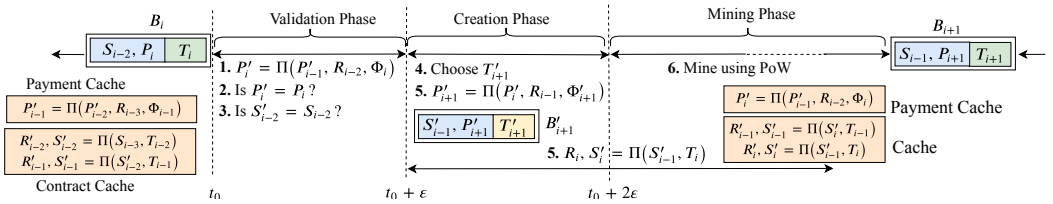


Fig. 7. Steps performed at a miner to validate a block B_i from the network in Tuxedo with $\zeta = 2$. After validation, miner creates the next potential block B'_{i+1} and starts PoW. During PoW for block B'_{i+1}, the miner computes the state required to instantly validate upcoming blocks.

execution of tx. Transactions with type1 are addressed to user accounts and transfers token1 from the from address to the to address. The amount of token1 transfer is present in the auxiliary information denoted using $\star$. Transactions of type2 are addressed to contract accounts and $\star$ contains the identity of the functions to be invoked and the required function call parameters. For every such transaction, miners in Tuxedo transfer an amount of token1, as a transaction fee from its sender account to a prespecified address denoted by Deposit. If in case tx.from does not have enough token1, a miner discards tx.

Two States. In our implementation of Tuxedo every miner maintains two different states: **P** and **S** where **P** is used to store information related to the amount of token1 in each user accounts and **S** is used to store the information regarding contract execution and amount of token2 in all the accounts. Since payment (including fees) and refund transactions only modify **P**, such a segregation enables faster validation and block creation.

Block Validation. Let $\mathbf{S}_{j-\zeta}$ and $\mathbf{P}_j$ be the contract and payment state at the end of block B_j. Also, $R_{j-\zeta}$ represents the refund processed after the execution of $T_{j-\zeta}$. Let the latest block known to an honest miner n_a be B_{i-1}. Let B_i be the next arriving block. Lets assume B_i is a non-ES block and n_a has already computed and cached the following state before its arrival.

$$\mathbf{P}'_{i-1} = \Pi(\mathbf{P}'_{i-2}, R_{i-\zeta-1} \| \Phi_{i-1}); \qquad R_{i-\zeta} \| \mathbf{S}'_{i-\zeta} = \Pi(\mathbf{S}'_{i-\zeta-1}, T_{i-\zeta}); \qquad R_{i-\zeta+1} \| \mathbf{S}'_{i-\zeta+1} = \Pi(\mathbf{S}'_{i-\zeta}, T_{i-\zeta+1})$$

Note that applying Π on any TOL T_j also outputs the ordered list of refund transactions corresponding to T_j. For $\zeta = 2$, this is depicted in Figure 7. On receiving the block B_i containing TOL T_i and digests of state $\mathbf{P}_i$ and $\mathbf{S}_{i-\zeta}$, n_a validates B_i as follows: (i) n_a first computes $\mathbf{P}'_i = \Pi(\mathbf{P}'_{i-1}, R_{i-\zeta} \| \Phi_i)$, (ii) checks whether $\mathbf{P}'_i$ matches with $\mathbf{P}_i$, and (iii) n_a also checks $\mathbf{S}'_{i-\zeta}$ matches $\mathbf{S}_{i-\zeta}$.

Alternatively, if n_a has not pre-computed $\mathbf{S}'_{i-\zeta}$ and B_i is a non-ES block, n_a continues to mine on the previous mining head till it computes $\mathbf{S}'_{i-\zeta}$ and re-starts validating B_i as above. However, if B_i is an ES-block and n_a has already validated the latest non-ES ancestor of B_i, n_a computes $\mathbf{P}'_i$ as $\Pi(\mathbf{P}'_{i-1}, \Phi_i)$ to check whether $\mathbf{P}'_i$ matches with $\mathbf{P}_i$ and skips step (iii) of validation. On successful validation, n_a accepts B_i and proceeds to create the next block as described below. Procedure Validate in Algorithm 1 presents the pseudo code for validation of blocks in Tuxedo.

Block Creation. On successful validation of the received block, to create the next block n_a (i) picks a new TOL T'_{i+1} from $\mathbb{T} \setminus T_i$, (ii) computes $\mathbf{P}'_{i+1} = \Pi(\mathbf{P}'_i, R_{i-\zeta+1} \| \Phi'_{i+1})$, (ii) fetches $\mathbf{S}'_{i-\zeta+1}$ from cache (if available), and (iii) creates the next potential block B'_{i+1} containing T'_{i+1} and digests of $\mathbf{P}'_{i+1}$ and $\mathbf{S}'_{i-\zeta+1}$. Also, the first non-ES block after a sequence of ES-blocks applies all accumulated refunds since the last non-ES block. Alternatively, if $\mathbf{S}'_{i-\zeta+1}$ is not available in the cache, n_a puts an *empty string* in place of $\mathbf{S}'_{i-\zeta+1}$. After creating B'_{i+1}, n_a immediately starts PoW on B'_{i+1}. Procedure Create in Algorithm 1 presents the pseudo code for validation of blocks in Tuxedo.

Execution of contract transactions In Tuxedo contract transactions are executed in parallel to PoW as shown in Figure 7. Specifically, during PoW for B'_{i+1}, n_a computes $R'_{i-\zeta}, \mathbf{S}'_{i-\zeta} = \Pi(\mathbf{S}'_{i-\zeta-1}, T_{i-\zeta})$. Also n_a adds T_i in the task queue (ref. §4.3) and executes T_i as soon as it executes all T_j for $j < i$, that appear prior to T_i.

6 ANALYSIS

We analyze the security of Tuxedo in the presence of a Byzantine adversary under all possible adversarial strategies. Two security properties of interest are *consistency* and *liveness* [15, 16]. Consistency (or k-consistency in [16]) requires that a block which orginated at least k time units in the past and is now present in the longest chain of any honest node will be present in every honest node's chain thereafter except with probability negligible in k. Liveness (or u-liveness in [16]) requires that the longest chain held by an honest node incorporates at least one fresh honest block over any period of u time units except with probability negligible in u.

Our analysis has the following outline. We start by modeling the blocks to be processed at an honest miner as a queuing system. Our queuing model captures the variability in the block inter-arrival time and the fact that honest miners might be processing old blocks as newer blocks continue to arrive. We then use our queuing analysis and the fact that honest miners are allowed to mine blocks with an empty state to illustrate that honest miners can extend blocks created by other honest miners even in a network with the worst possible latency. This implies that the average growth of the longest chain at any honest node, or *chain growth*, in Tuxedo is greater than or equal to the chain growth of PoW based Nakamoto consensus. We then leverage results from Gazi et al. and Dembo et al. [12, 16] for PoW consensus that show that when an adversary mining power is strictly less than the chain growth, then a chain with PoW consensus has both k-consistency and u-liveness. Since Tuxedo retains the same chain growth as Nakamoto consensus, the same results are easily extended to Tuxedo as well.

6.1 Block Processing as a Queuing System

The arrival of blocks in PoW blockchain can be modeled as a Poisson process with arrival rate λ, or equivalently $1/\lambda$ is the expected inter-arrival time between two consecutive blocks [25]. As all honest miners take τ units of time to process a block, i.e. the processing rate of the server is $1/\tau$. On arrival of every new block B_i with TOL T_i that extends a chain longer than the current mining head at a miner n_a, n_a adds the block to its queue. n_a processes (that is, validates) these blocks in First In First Out (FIFO) order. As we have mentioned earlier, due to forks, there will be multiple queues at each miner (see Figure 6), but our analysis applies to any of them as arrival rate at each queue is dominated by the arrival rate in a single queue setting and a miner processes all queues in parallel.

Let $Q_a(t)$ denote the size of the queue of a miner n_a at time t. If block B_k enters the queue at time t_k, we use $Q_a(t_k^-)$ and $Q_a(t_k^+)$ to denote the size of queue immediately before and after time t_k respectively. Note, $Q_a(t_k^+) = Q_a(t_k^-) + 1$.

Handling non-ES blocks. The ability miner n_a to validate a received block B_k and/or create an non-ES block on it is directly related to the number of blocks in the queue which B_k enters. Notice that if $Q_a(t_k^+) > \zeta$ then the head of the queue contains TOL T_i for $i \leq k - \zeta$, and the miner will not be able immediately validate validate B_k. Similarly, when $Q(t^+) = \zeta$, the miner will be able to validate B_k but will not have the state to immediately mine a non-ES block on top of the received block.

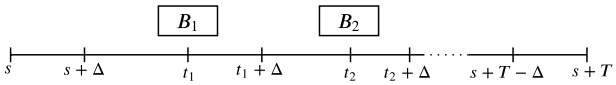


Fig. 8. Honest blocks chosen in time interval $[s+\Delta, s+T-\Delta]$ where the chosen blocks are separated by at least a gap of Δ.

6.2 Reduction of Tuxedo to Nakamoto PoW

In this section we illustrate that the *chain-growth* of Tuxedo (as defined in [19, 27, 28]) in a time interval T is greater than or equal to the known chain growth of PoW based Nakamoto consensus [19, 27, 28]. As mentioned earlier, we later use this fact to prove security of Tuxedo against all possible Byzantine adversaries.

Consider a time interval $[s, s+T]$. Select the honest blocks as shown in Figure 8. First skip ahead to $s+\Delta$, then find the next honest block, then skip by Δ and then repeat till time $s+T-\Delta$. Let's call these blocks $B_1, B_2, \cdots, B_N$. Let B_k be the honest block generated at time t_k.

Lemma 6.1. *Let B_k be the m^{th} block starting from genesis block in the chain containing B_k. Call these blocks $b_0, b_1, b_2, \cdots, b_m$ with b_0 as the genesis block. Let $t'_0, t'_1, t'_2, \cdots, t'_m$ be the time when an honest miner hears the corresponding block for the first time. Then by $t_k+\Delta$ i.e. $t'_m+\Delta$, all honest miners would have processed the state required to validate B_k.*

Proof. Let Q_L and Q_U be two hypothetical FIFO queues with constant service rate $1/\tau$ in which blocks $b_1, b_2, \cdots, b_m$ enter at $t'_1, t'_2, \cdots, t'_m$ and $t'_1+\Delta, t'_2+\Delta, \cdots, t'_m+\Delta$ respectively. Let $Q_a^{(k)}(t)$ be the position of the block B_k (i.e. b_m) at the queue of miner n_a at time t, and $Q_{B_k}(t)$ be its position at the miner which created it. Then the following two conditions hold:

$$Q_L^{(k)}(t) \leq Q_a^{(k)}(t), \forall t \text{ after } B_k \text{ enters } Q_a \qquad \text{and} \qquad Q_a^{(k)}(t) \leq Q_U^{(k)}(t), \forall t \geq t_k+\Delta \tag{2}$$

Hence,

$$Q_a^{(k)}(t_k+\Delta) \leq Q_U^{(k)}(t_k+\Delta) = Q_L^{(k)}(t_k) \leq Q_{B_k}^{(k)}(t_k) \tag{3}$$

Equation (3) implies that by time $t_k+\Delta$, the position of B_k in the queue of all honest miners will be less than or equal to the position of B_k in the queue its creator at time t_k. Hence if $Q_{B_k}(t_k^+) \leq \zeta$, i.e. B_k is a non-ES block, by time $t_k+\Delta$, the position B_k will be lower than or equal to ζ at all honest miners and hence, all honest miners will be able to validate B_k by time $t_k+\Delta$. □

Lemma 6.1 implies that whenever an honest miner generates a block, Δ time after the block generation time, every honest miner will have the state required to validate the block generated by the honest miner. Hence, Δ time after generation of an honest block B_k, every honest miner extends a block which is at a height greater than or equal to the height of B_k. We use this argument in Lemma 6.2 to prove that whenever two honest miners generate blocks at time instants that are at least Δ apart, the height of the latter block is greater than the height of the former. As a result, it is easy to see that blocks $B_1, B_2, \cdots, B_N$ that we consider have a strictly increasing height.

Lemma 6.2. *Let $\ell(B_k)$ denote the length of the block starting from genesis block b_0 with $\ell(b_0)=0$. Then B_i for all $i \in [1, N]$ which were mined between $[s+\Delta, s+T-\Delta]$ as shown in Figure 8 have distinct length. Further, $\ell(B_i) > \ell(B_j), \forall i > j$.*

Proof. Consider two consecutive blocks B_k, B_{k+1} mined at time t_k, t_{k+1} respectively (need not be part of the same chain). Let n_k, n_{k+1} be the miners of B_k, B_{k+1} respectively. Since $t_{k+1} > t_k+\Delta$, n_{k+1} would have heard of B_k prior mining B_{k+1}. Also, from Lemma 6.1 by time $t_k+\Delta$, n_{k+1} will have the state to validate B_k. Thus from time $t_k+\Delta$ onwards, n_{k+1} either will extend B_k or any other validated

block with same or greater length than B_k. This implies $\ell(B_{k+1}) > \ell(B_k)$. This is true for all pair of consecutive blocks and hence by transitivity of length comparison, we get $\ell(B_i) > \ell(B_j), \forall i > j$. □

Next we will use the Lemma 6.1 and Lemma 6.2 show that during an interval of size T, the height of the blockchain at every honest miner grows by at least N.

Lemma 6.3. (Chain Increase) *Let $L_j(t)$ be the length of the longest validated chain at miner n_j at time t. Let $L_{\min}(t), L_{\max}(t)$ be the minimum and maximum of chain lengths of all honest miners at any time t, i.e. $L_{\min}(t) = \min_j\{L_j(t)\}$ and $L_{\max}(t) = \max_j\{L_j(t)\}$, then in the scenario show in Figure 8, chain length of all honest miners grows by at least N blocks, i.e $L_{\min}(s+T) \geq L_{\max}(s) + N$.*

Proof. From Lemma 6.2, $\ell(B_{i+1}) \geq \ell(B_i) + 1, \forall\, i \in [N-1]$. Let $L_{B_k}(t_k^+)$ be the length of the longest chain of miner of block B_k at time t_k^+. Then from Lemma 6.2 we know, $L_{B_{k+1}}(t_{k+1}^+) \geq L_{B_k}(t_k^+) + 1$. From Lemma 6.1, $L_{\min}(t_k^+ + \Delta) \geq L_{B_k}(t_k^+)$. Also, $L_{\min}(s+\Delta) \geq L_{\max}(s)$ as all blocks generated before s reaches every honest miner by time $s + \Delta$. Hence,

$$L_{\min}(s+T) \geq L_{\min}(t_N + \Delta) \geq L_{\min}(s+\Delta) + N \geq L_{\max}(s) + N$$ □

Notice that the scenario shown in Figure 8 is stochastically identical to the HYB experiment shown by Pass et al. in [27]. Hence the chain growth of an honest miner for a PoW chain using Tuxedo is the same as chain growth for Bitcoin presented in [27]. We reproduce the main result from [16] below for completeness. Let p_a and p_h be the adversarial and honest probabilites of successfully mining a block in a single slot (of size δ time), and the chain growth per slot is given[4] by $\vartheta(p_h, \Delta) := \frac{1}{\Delta/\delta - 1 + 1/p_h}$.

Theorem 6.4. *(Theorem 16 in [16]). If $p_a < \vartheta(p_h, \Delta)$ then a Bitcoin execution over a lifetime of L slots achieves k-consistency and u-liveness except with error probabilities $(L\delta/\Delta)\exp(\Omega(k))$ and $(L\delta/\Delta)\exp(\Omega(u)$, respectively. If $p_a > \vartheta(p_h, \Delta)$, then the private chain attack[5] is successful (with probability tending to 1 exponentially quickly), and Bitcoin is insecure.*

The proof of the above theorem extends trivially [6] to Tuxedo which we have shown has the same chain growth as a chain with Nakamoto consensus.

6.3 Choice of ζ

As $Q(t) \geq \zeta$ at a honest miner implies that the miner will not be able to validate a received non-ES block, we compute an upper bound on $\Pr[Q(t) \geq \zeta]$ under all possible adversarial strategies after making certain approximations. Recall that $Q(t) \geq \zeta$ do not violate security of Tuxedo and hence the guarantees provided in previous section still holds true.

A well-known result from queuing theory [26] is that in any queuing system with constant service rate $1/\tau$, the size of the queue at any time t is given by:

$$Q(t) = \sup_s \left\{ A(s) - \frac{s}{\tau} \right\}, \tag{4}$$

where $A(s)$ is the number of arrivals during the interval $[t-s, t]$. In addition to the number of blocks generated during the time interval $[t-s, t]$, $A(s)$ may also include honest blocks from time interval $[t-s-\Delta, t-s]$ as these blocks might be delayed due to network. Furthermore, an adversary can

[4] Δ in our work corresponds to Δ_0 in [16].

[5] The private chain attack is an attack in which the adversary mines a chain privately starting from the parent of a block it wants to orphan, and releases the chain when it becomes longer than the longest honest chain.

[6] Earlier works provided loose bounds on maximum adversary power to ensure consistency, using concepts such as "convergence opportunities" [19, 27]. However, recent work [16] gives a tight bound without using convergence opportunities. The proof relies mainly on chain growth, which we have shown is the same for Tuxedo as in Nakamoto consensus.

deliberately withhold blocks mined prior to time $t-s$ and release them during $[t-s, t]$. However, as demonstrated in [27] that to withhold a block by longer than time t_w, adversary needs to generate a private chain longer than honest chain during that time.

Approximating the growth rate of the honest miners as a Poisson process with rate γ where $\gamma = \alpha/(1+\Delta\alpha)$, we can approximate the race between honest chain and the adversarial chain for time t_w as a Skellam Distribution [31] with $\mu_1 = \beta t_w$ and $\mu_2 = \gamma t_w$. Specifically, let $N(t_w), X_{\mathcal{A}}(t_w)$ be the random variables denoting the chain growth of honest miner and number of blocks generated by $\mathcal{A}$ during a time interval of size t_w respectively. Then the success probability of $\mathcal{A}$ withholding a block for longer than t_w is $\Pr[X_{\mathcal{A}}(t_w) - N(t_w) > 0]$. Since, $X_{\mathcal{A}}(t_w)$ and $N(t_w)$ are independent Poisson random variable, $X_{\mathcal{A}}(t_w) - N(t_w)$ follows a Skellam distribution with mean μ_1 and μ_2 as mentioned above.

Using results from Skellam distribution, given a small threshold η, we pick a value of t^* such that

$$\Pr[X_{\mathcal{A}}(t^*) - Y(t^*)] \leq \eta \tag{5}$$

and assume that $\mathcal{A}$ is not allowed to withhold a block for more than t^* units of time. Under this assumption, we next upper bound the probability that queue of an honest miner will exceed any given ζ under all possible adversarial strategies.

Theorem 6.5. *For any given $\epsilon_0, \epsilon_1, t^*$, let $s_0 = \max\{\frac{\Delta}{\epsilon_0}, \frac{t^*}{\epsilon_1}\}$ and $\bar{\lambda} = (1+\epsilon_0)\alpha + (1+\epsilon_1)\beta$. Let $Q(t)$ be the size of an honest miner's queue at time t. Then*

$$\Pr[Q(t) \geq \zeta] \leq \sum_{i=\zeta}^{\infty} \pi_i + 1 - \sum_{i=0}^{\zeta-1} \frac{\bar{\lambda}^i e^{-\bar{\lambda}s_0}}{i!}, \tag{6}$$

where π_i is the stationary distribution of M/D/1 queue with arrival rate $\bar{\lambda}$.

Using our worst-cast analysis, we suggest concrete values of ζ one should consider to bound the probability of an honest miner's queue exceeding ζ. As we expect attacks to be intermittent (if any), we also numerically compute these bounds for an honest execution of the protocol, i.e., in a network without any adversary. Figure 19 and 18 plot the result of Theorem 6.5 under some example parameters.

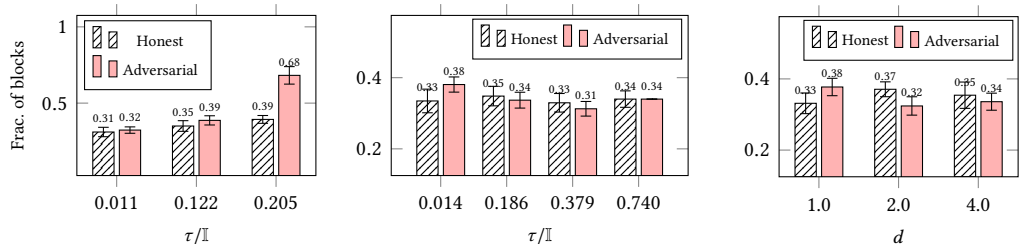
Remark. It is important to note the event $Q(t) \geq \zeta$ does not let the Byzantine adversary to violate the consistency of the protocol. Instead, it only allows the adversary to delay the reporting of update state for a very short duration of time. This is because, when $Q(t) \geq \zeta$ honest miners do not extend blocks mined by the adversary, and since the block processing rate is considerably higher than the block generation rate of the adversary, the queue at the honest miners will soon have less than ζ blocks. Hence, very soon the honest miners will start creating blocks with non-empty state.

7 EVALUATION

We implement all parts of Tuxedo along with an on-demand adversarial behavior o skip validation of transactions atop Ethereum Geth client version 1.9.3. We then compare the performance of Tuxedo with that of Ethereum. To facilitate such comparisons, we implement an adversary who skips validation and/or creation of blocks in Ethereum.

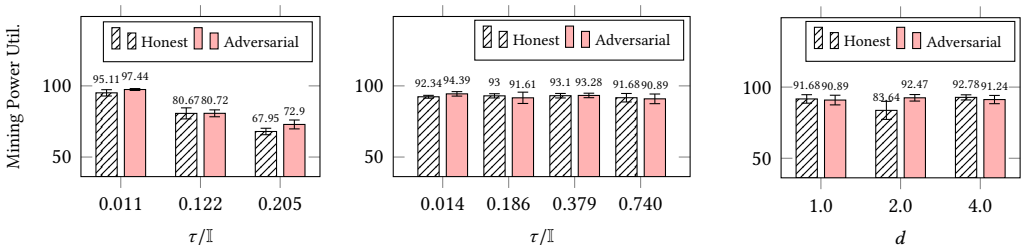
7.1 Experimental Setup

Our experimental setup consists of 50 virtual machines (VMs) running in Oracle Cloud. All expect one of the VMs are dual-core machines with 8GM of RAM running Ubuntu.16.04. The remaining VM has 8 core CPU @2.19 GHz and 30GB, and we use it as an adversarial miner. We deliberately assign one miner a computational advantage over others to measure its effects on the fairness



(a) Ethereum, $d = 1$ (b) TUXEDO, $d = 1$ (c) TUXEDO, $\tau/\mathbb{I} = 0.74$

Fig. 9. Fraction of blocks mined by the miner m_1 controlling 33% mining power and $c = 0.6$ in Ethereum and TUXEDO



(a) Ethereum, $d = 1$ (b) TUXEDO, $d = 1$ (c) TUXEDO, $\tau/\mathbb{I} = 0.74$

Fig. 10. Mining power utilization of the (a)Ethereum and (b),(c) TUXEDO with increasing $\tau/\mathbb{I}$ and Network delay.

of Ethereum and TUXEDO. In our setup, all miners have identical network bandwidth of 1GB/s download and 100MB/s upload speeds.

Node. Each VM in our experimental setup runs one TUXEDO miner. The mining power of each miner is set according to the distribution of top 50 Ethereum miners extracted from [2]. This corresponds to 99.98% of Ethereum's total mining power. For each miner in our setup, we simulate its block mining process by drawing the interarrival time between the blocks from an exponential distribution with parameter λ/h where h is the fraction of mining power controlled by the node. Table 1 reports the percentage of mining power controlled by the top 14 miners that sum up to 97% of the total mining power.

32.98	16.16	15.06	5.72	5.67	4.41	4.14
3.53	2.61	1.84	1.34	1.32	1.25	1.05

Table 1. Percentage of mining power controlled by top 14 miners in descending order in our experimental setup. For e.g. first and 14th miner controls 32.98% and 1.05% of the mining power, respectively.

Network. To make a prototype of an Ethereum mining network, we collected data about the geographical location of the top 50 miners in Ethereum. Each miner in our experiment emulates the geographical location of one such miner. We then form a randomly connected network of these miners where the degree distribution follows a power law with exponent −2.5. Communication delays between every pair of miners in our network are set accordingly to the ping delays observed between respective geographical locations [1]. We use Linux `tc` command to simulate the link delays.

Methodology. We test Tuxedo by deploying 50 contracts, each implementing Quicksort, 2D matrix multiplication, and iteration with basic arithmetic operations. We then invoke functions from each contract with appropriate parameters to achieve the desired block processing time. Throughout our experiment, we ensure that each block contains ~165 transactions in total, which is roughly the average number of transactions in an Ethereum block. As we simulate an adversary ($\mathcal{A}$) that skips validation of blocks and creates new ones with contracts whose execution results are already known to the $\mathcal{A}$, we deliberately restrict all of the above-mentioned contracts to be stateless. Note that, as the primary metric of evaluating Tuxedo is the processing time of a block, any choice of contracts will give us the same results as long as they achieve the desired block processing time.

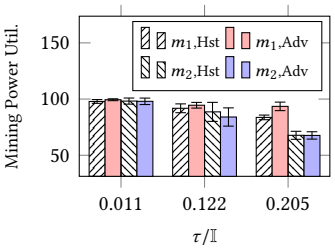


Fig. 11. Mining power utilization of first two miner (m_1 and m_2), Ethereum, $d = 1$

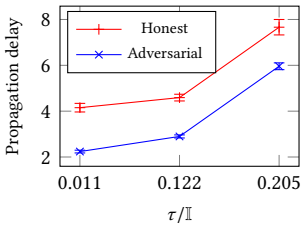


Fig. 12. Measured median propagation delay of blocks with increasing $\tau/\mathbb{I}$ in Ethereum.

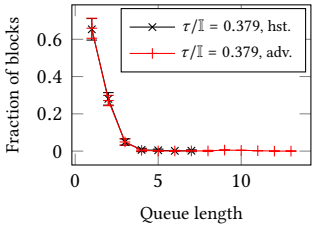


Fig. 13. Average queue size of honest miners (excluding m_1) for $\tau/\mathbb{I}$ = 0.379 and $d = 1$

7.2 Experiments and Results

We first evaluate the effect of increasing $\tau/\mathbb{I}$ in Ethereum with all miners being honest. We then repeat the experiment in the presence of an adversarial miner, who skips validation of received blocks and creates blocks with transactions for which the miner already knows the execution results. We then perform the same set of experiments with Tuxedo and compare our results with Ethereum. Next for fixed $\tau/\mathbb{I} = 0.70$ we evaluate Tuxedo with increasing network delay.

In all experiments, the first miner (m_1) controls ~33% of the network's mining power and can process transactions ~1.67 times faster than other miners. Also, we keep block mining difficulty such that $1/\lambda = 15.0$. Hence $\mathbb{I}$ is equal $2\tau + 1/\lambda$ for Ethereum experiments and $1/\lambda$ for Tuxedo experiments.

Fairness violations in Ethereum. Figure 9a illustrates the fraction of Ethereum blocks mined by m_1. Observe that with increasing $\tau/\mathbb{I}$, the fraction of blocks m_1 mines increases even when m_1 is honest. This corroborates our theoretical analysis in Appendix C. When m_1 is adversarial and skips both block validation and block creation, m_1 mines a significantly higher fraction of blocks. These fractions are higher than the theoretically computed fraction in Figure 9a. Because unlike our simplistic assumption in theoretical analysis (ref.C), in the experiments m_1 mines for the entire duration of the experiment.

High fork rate in Ethereum. With high $\tau/\mathbb{I}$, we observe that the fork rate of Ethereum increases. We use *Mining Power Utilization* (MPU) of a blockchain be the fraction of blocks mined by the miners that end up in the eventual longest chain. A blockchain with lower MPU implies that a lower fraction of mined blocks ends up being in the blockchain and that many blocks are orphaned, i.e., the fork rate is higher. Figure 10a illustrates the MPU of Ethereum with increasing $\tau/\mathbb{I}$. Notice that despite having $\lambda = 1/15$ and $\mathbb{I} = 2\tau + 1/\lambda$, the fork rate increases. This is because miners in Ethereum only forward a block when they have fully validated its parent block. Thus with high $\tau/\mathbb{I}$,

miners in Ethereum more frequently encounter blocks whose parents are yet to be validated by the miners. As a result, with increasing $\tau/\mathbb{I}$, the effective propagation delay of blocks in the network increases, which leads to higher forks and lowers the MPU of the network.

Figure 12 illustrates that the median propagation delay of our Ethereum network. Median block propagation delay in the presence of an adversarial node is lower due to the fact that the adversarial node can forward blocks immediately as it processes all received blocks immediately. Also, we observe that higher $\tau/\mathbb{I}$ does not lower MPU of the adversarial miner m_1 by much. Furthermore, MPU of an adversarial miner is much higher than its honest counterpart. Figure 11 illustrates the mining power utilization of top 2 miners in our experimental setup (ref. 1). Note that when m_1 is adversarial, it mines solo for a longer duration. This allows m_1 to mine longer sequence of blocks more frequently, while other miners were busy extending an older block. Hence, when the network gets synchronized again, honest nodes have to backlog of blocks to process, the string of blocks mined by the m_1 enters the blockchain with high probability.

Increasing $\tau/\mathbb{I}$ in Tuxedo. We observe that higher $\tau/\mathbb{I}$ does not affect miners in Tuxedo by repeating the above experiments in Tuxedo. Figure 9b illustrates that the fraction of blocks m_1 mines does not vary in Tuxedo. This holds despite high $\tau/\mathbb{I}$ and an adversarial m_1. Also, unlike Ethereum, high $\tau/\mathbb{I}$ does not affect the mining power utilization of Tuxedo since all miners can immediately validate all received blocks. We illustrate this in Figure 10b. We use ζ based on our analysis in §6.3 for $t^* = 0$ and $\Pr[Q(t) \geq \zeta] \leq 2^{-15}$.

Figure 13 demonstrates the queue observed by a arriving block in an honest miner for $\tau/\mathbb{I} = 0.379$. Observe that presence of a skipping adversary does alter the queue size honest miners observe. Furthermore, more than 99% of the blocks find a queue size of four or less, and less than $< 0.1\%$ blocks find a queue size of eight or higher. Thus, though ζ is chosen to be high, for a majority of the blocks, the users of Tuxedo will get the execution results of their transactions within four blocks.

Increasing network delay. For fixed $\tau/\mathbb{I} = 0.74$ we evaluate Tuxedo with increasing delay. Specifically, we increase link delay between each pair of connected nodes by a factor of $d = 1, 2, 4$. Figure 9c illustrates that higher delay does not affect the fraction of blocks mined by m_1 for both honest and adversarial m_1. Additionally, we observe that despite higher network delay, the MPU of Tuxedo does not decrease in (see Figure 10c). This is because the top five miners controlling approximately 75% of mining power in our experimental network are well connected, often directly connected, with each other. Hence the increased delay does not affect the block propagation delay between them.

8 RELATED WORK

There have been attempts to enable the execution of computationally intensive smart contracts [9, 11, 14, 18, 32, 36] through off-chain solutions where the execution of intensive transactions is delegated to a subset of miners or volunteer nodes. These solutions typically have long latency for off-chain computations and make additional security assumptions beyond those required for PoW consensus. Also, off-chain solutions restrict certain interactions between contracts, e.g., one smart contract cannot internally invoke functions of other smart contracts. Such interactions are desirable and often occur in practice, as can be seen from figure 1b.

Arbitrum [18] requires nodes participating in the protocol to be rational and one participating node to be honest. Yoda [11] requires an unbiased source of randomness. Ekiden [9] relies on SGX enclaves and requires all enclaves to be trusted, an assumption that is made questionable by recent attacks [8, 34]. In Zokrates [14], participants are required to generate expensive non-interactive proofs for verification of off-chain computations. Although ACE [36] supports interactive calls between smart contracts, those transactions have very long delays as they can be delayed by the

slowest committee involved in the interaction. Also, ACE relies on a reputation-based mechanism to realize such assumptions.

Note that Tuxedo does not eliminate the benefits of off-chain based approaches. Instead, it further opens up the possibilities for more comprehensive (computationally intensive, if needed) on-chain aggregation schemes or off-chain schemes such as verifiable computation [21]. This can further scale the overall computation capacity of the entire system.

In an alternate approach, a recent line of work has tried to increase the scalability of smart contracts by concurrently executing transactions [6, 13, 38]. Dickerson et al. [13] enables miners to concurrently execute transactions using a pessimistic abstract lock and inverse-log represented as a directed acyclic graph (happen-before graph). This inverse-log is later used in the validation phase to replay the block creator's parallelization schedule. Anjana et al. [6] replaced the pessimistic lock with Optimistic Concurrency Control favoring low-conflict workloads but at the cost of a high abort rate for transactions with more conflicts. Zhang et al. [38] improves concurrency of the validation phase by recording the write set of each transaction in the block, which incurs additional storage and communication overhead. These works are optimistic because the adversary can always create blocks whose validation cannot be parallelized. Hence, they are not suitable block validation time in the presence of a Byzantine adversary.

Blockchain sharding based solutions [5, 20, 22, 35, 37] partition the set of nodes in the system into smaller groups, known as *shards*, where each shard maintains a subset of blockchain states and processes a subset of transactions. They scale the overall transaction processing capacity of the system in proportion to the number of shards. However, the processing capacity of each shard is still limited, and hence they are not suitable for executing computationally intensive transactions within each shard. Additionally, cross-shard transactions, i.e., transactions that modify state from more than one shard, still suffer from longer latency [30]. It is clear that in a PoW sharded blockchain system with no cross-shard transactions, Tuxedo can be used in each shard to increase its processing capacity. However, extending Tuxedo to cases with cross-shard transactions raises several technical challenges, which we will address in future work.

9 CONCLUSION

We have presented Tuxedo which theoretically allows validation time of blocks in PoW based blockchains to be comparable to the average interarrival time, i.e., $\tau/\mathbb{I} \approx 1$. Hence, it makes blockchains accessible to smart contracts that require heavy computation.

Another advantage of the on-chain approach is that all miners update states locally. Hence, unlike off-chain solutions, Tuxedo obviates the need to transfer state updated by one set of miners to another. Furthermore, Tuxedo does not introduce any extra message exchange or requirement of trust between nodes. We prove the security of Tuxedo in a synchronous network with an end-to-end delay of Δ in the presence of a Byzantine adversary considering all possible adversarial strategies. We also present a principled approach to select *ths* for any given choice of parameters.

While we have introduced Tuxedo for a single PoW blockchain, we believe Tuxedo has the potential to increase computation in blockchains using other non-PoW consensus protocols as well as in sharded blockchains. Exploring these issues is part of future work.

ACKNOWLEDGMENTS

The authors would like to thank Andrew Miller, Manoj M. Prabhakaran, and anonymous reviewer for their feedback on the paper. The work is supported in part by a generous grant of cloud credits from Oracle Corp., that we have used to run all of the experiments whose results are reported in this paper.

REFERENCES

[1] [n.d.]. Global Ping Latency. https://investoon.com/mining_pools/eth [Online; accessed 16-May-2019].

[2] [n.d.]. Mining Power Distribution of Ethereum. https://investoon.com/charts/mining/eth [Online; accessed 16-May-2019].

[3] [n.d.]. Tornado.Cash. https://tornado.cash/

[4] [n.d.]. ZK-Rollups. https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/

[5] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. 2018. Chainspace: A Sharded Smart Contract Platform. In *Network and Distributed System Security Symposium 2018 (NDSS 2018)*.

[6] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. 2018. An Efficient Framework for Concurrent Execution of Smart Contracts. *CoRR* abs/1809.01326 (2018). arXiv:1809.01326 http://arxiv.org/abs/1809.01326

[7] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 947–964.

[8] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*.

[9] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 185–200.

[10] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. 2018. Edrax: A Cryptocurrency with Stateless Transaction Validation. (2018).

[11] Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. 2019. YODA: Enabling computationally intensive contracts on blockchains with Byzantine and Selfish nodes. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*.

[12] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2020. Everything is a race and Nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 859–878.

[13] Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. 2017. Adding Concurrency to Smart Contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC '17)*. 303–312.

[14] Jacob Eberhardt and Stefan Tai. 2018. ZoKrates-Scalable Privacy-Preserving Off-Chain Computations. In *IEEE International Conference on Blockchain*.

[15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 281–310.

[16] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2020. Tight consistency bounds for bitcoin. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 819–838.

[17] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. 2018. Decentralization in bitcoin and ethereum networks. In *International Conference on Financial Cryptography and Data Security*. Springer, 439–457.

[18] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1353–1370.

[19] Lucianna Kiffer, Rajmohan Rajaraman, et al. 2018. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 729–744.

[20] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[21] Jonathan Lee, Kirill Nikitin, and Srinath Setty. 2020. Replicated state machines without replicated execution. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 119–134.

[22] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 17–30.

[23] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 706–719.

[24] Ujan Mukhopadhyay, Anthony Skjellum, Oluwakemi Hambolu, Jon Oakley, Lu Yu, and Richard Brooks. 2016. A brief survey of Cryptocurrency systems. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. 745–752. https://doi.org/10.1109/PST.2016.7906988

[25] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[26] Ilkka Norros. 1994. A storage model with self-similar input. *Queueing systems* 16, 3-4 (1994), 387–396.

[27] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 643–673.
[28] Ling Ren. 2019. Analysis of Nakamoto Consensus. Cryptology ePrint Archive, Report 2019/943. https://eprint.iacr.org/2019/943.
[29] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. 2018. TLS-N: non-repudiation over TLS enabling ubiquitous content signing for disintermediation. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium*.
[30] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. 2021. Blockchains vs. Distributed Databases: Dichotomy and Fusion. In *Proceedings of the 2021 International Conference on Management of Data*. 1504–1517.
[31] John G Skellam. 1946. The frequency distribution of the difference between two Poisson variates belonging to different populations. *Journal of the Royal Statistical Society. Series A (General)* 109, Pt 3 (1946), 296–296.
[32] Jason Teutsch and Christian Reitwießner. 2017. A scalable verification solution for blockchains. (2017).
[33] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. 2020. Aggregatable subvector commitments for stateless cryptocurrencies. In *International Conference on Security and Cryptography for Networks*. Springer, 45–64.
[34] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 991–1008.
[35] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 95–112.
[36] Karl Wüst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. 2020. ACE: Asynchronous and concurrent execution of complex smart contracts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 587–600.
[37] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 931–948.
[38] An Zhang and Kunlong Zhang. 2018. Enabling concurrency on smart contracts using multiversion ordering. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*. Springer, 425–439.

A DISCUSSION

Affect of network delay in Fork-rate. Both in theory and practice, an increase in network delay affects Tuxedo with a large validation time identically to as it affects existing Ethereum with low validation time. More specifically, our theoretical analysis of security (Theorem 6.4) matches the known proven security results of Ethereum/PoW blockchains, i.e., the network delay term Δ identically affects theoretical bounds. Furthermore, this is independent of the validation time τ. The same holds in practice as well. More concretely, Figure 12 illustrates that more than 99% of the time, the backlog queue at honest miners is less than 5, which is much less than ζ.

Hence, during block propagation of a new block, every miner will be able to immediately perform the required preliminary checks and forward the block to others. Since the time it takes for a miner to perform the preliminary checks is similar to the current block validation time in Ethereum, the overall block propagation delay is similar to that of the existing Ethereum propagation delay with a small block validation time.

Handling invalid transactions. Invalid transactions in Tuxedo such as, runs out of gas, are treated in the same way as Ethereum. Since the execution of transactions is deterministic, if a transaction fails at a node, it will fail at every other node. So each node will locally immediately roll back the states modified by the failed transaction and confiscate its transaction fee.

Consequences of picking a smaller ζ. A smaller ζ do not let a Byzantine adversary violate consistency of the protocol. Instead, in the worst case it only allows the adversary to delay reporting of the updated state for a very short duration of time. When $Q(t) \geq \zeta$ honest miners do not extend blocks mined by the adversary, and since the block processing rate is considerably higher than the block generation rate of the adversary, the queue at the honest miners will soon have less than

ζ blocks. Hence, very soon the honest miners will start creating blocks with non-empty state. In practice, we observe that $\zeta = 5$ enables miners to successfully report final state for more than 99% of the blocks.

Light clients in Tuxedo. Blocks in Tuxedo maintain the cryptographic digest of the state, which allows light clients to efficiently validate or prove to other light clients, a portion of the latest state similar to Ethereum.

Practical Latency. Although, state corresponding to a contract transactions gets reported ζ block later, our analysis and evaluation demonstrate that, most ($> 99\%$) blocks in Tuxedo finds a queue size of less than **five** on its arrival. Hence in practice, miners will have execution results of transactions reasonably quickly. Furthermore, payment transactions are executed immediately thus can be used in the low latency applications. Our experimental results demonstrate working of Tuxedo for $\tau/\mathbb{I} = 0.70$ for an implementation over a standard Ethereum geth client.

Comparison with Rollups [4] Rollups are a Layer 2 solution that require "operators" to stake a bond in the rollup contract. This is not proposed as the smart contract scaling solution but instead a high throughput solution that enhances the transaction throughput between the operators involved in the bond. It also fails in the case of interactive smart contract transactions.

Mitigating malicious clients from filling blocks with junk transactions Since, miners in Tuxedo seal blocks without first executing their computationally intensive transactions, we need to prevent clients from filling blocks with junk transactions. As we discuss in §5 and 4.4, each computationally intensive transaction needs to pay a fee that specifies the maximum amount of time (in terms of gas) the transaction will consume. This fees are deposited as soon as transactions are included in the chain. Also, for every such transactions, the miner collects some fees while executing the transactions. Thus, a client needs to pay a high transaction fees to fill the block with junk transactions. We believe that in practice such attacks will be rare and ephemeral.

Incentives to include non-zero state. Similar to Ethereum, in Tuxedo we can award the transaction fees for computationally intensive transactions to the miner that report its state. This will incentivize the miners to include non-empty state in the block. We briefly mention this in §5 leave its detailed design and analysis as a future work.

Tuxedo and other consensus protocols. Tuxedo's idea of parallelizing execution of transactions and participation in consensus protocol can be applied to other consensus protocols such as Algorand, proof-of-stake, and BFT protocols. Determining the gains it provides with these protocols is a potential future research direction.

Algorithm 1 TUXEDO for $\zeta \geq 2$

$\mathbb{T} : \{tx_j | j = 1, 2, \cdots\}$ ▷ Transaction pool at the miner
$\mathbb{C} : \left\{R'_j, \mathbf{S}'_j = \Pi(\mathbf{S}'_{j-1}, T_j)\right\}$ ▷ Contract Cache
$\mathbb{P} : \left\{\mathbf{P}'_j = \Pi(\mathbf{P}'_{j-1}, R_{j-\zeta}, \Phi_j)\right\}$ ▷ Payment Cache
$\mathbb{Q} : \{T_j | j = 1, 2, \cdots\}$ ▷ TOL that are yet to be processed
ProcessTOL(·) ▷ Non-blocking call to process existing TOL
while true **do**
    Reconfigure(B_k) ▷ On arrival of new block
procedure Reconfigure(B_k)
    if Validate(B_k) **then**
        stop current PoW
        $B'_{k+1} \leftarrow$ Create(B_k)
        start PoW on B'_{k+1}
procedure Validate(B_k)
    $valid \leftarrow$ **false**; $\mathbf{S}_{k-\zeta}, \mathbf{P}_k, T_k \leftarrow B_k$
    if $\mathbf{S}_{k-\zeta}$ is empty **then**
        $\mathbf{P}'_k \leftarrow \Pi(\mathbf{P}'_{k-1}, \Phi_k)$
        if $\mathbf{P}'_k = \mathbf{P}_k$ **then**
            $valid \leftarrow$ **true**
    else
        if $\mathbf{S}'_{k-\zeta}$ not in cache **then**
            $valid \leftarrow$ **false**; add T_k to $\mathbb{Q}$
        else
            $\mathbf{P}'_k \leftarrow \Pi(\mathbf{P}'_{k-1}, R'_{k-\zeta}, \Phi_k)$
            if $\mathbf{P}'_k = \mathbf{P}_k$ **and** $\mathbf{S}'_{k-\zeta} = \mathbf{S}_{k-\zeta}$ **then**
                add $\mathbf{P}_k$ to $\mathbb{P}$; add T_k to $\mathbb{Q}$; $\mathbb{T} \leftarrow \mathbb{T} \setminus T_k$
                $valid \leftarrow$ **true**
    return $valid$
procedure Create(B_k)
    $T'_{k+1} \leftarrow$ subset of $\mathbb{T}$
    if $\mathbf{S}'_{k+1-\zeta}$ in cache **then**
        $\mathbf{P}'_{k+1} \leftarrow \Pi(\mathbf{P}'_k, R'_{k+1-\zeta}, \Phi'_{k+1})$
        return $(\mathbf{S}'_{k+1-\zeta}, \mathbf{P}'_{k+1}, T'_{k+1})$
    else
        $\mathbf{P}'_{k+1} \leftarrow \Pi(\mathbf{P}'_k, \Phi'_{k+1})$
        return $(\textit{empty-string}, \mathbf{P}'_{k+1}, T'_{k+1})$
procedure ProcessTOL(·)
    while true **do**
        if $\mathbb{Q}$ is non empty **then**
            $B_j \leftarrow$ next block in $\mathbb{Q}$
            $T_j \leftarrow$ TOL of B_j
            $R_j, \mathbf{S}_j = \Pi(\mathbf{S}_{j-1}, T_j)$
            add $R_j, \mathbf{S}_j$ to $\mathbb{C}$
            if $j >$ current validated chain length **then**
                Non-blocking Reconfigure(B_j)

B PROOF OF THEOREM 2

Let $X_{\mathcal{H}}(b), X_{\mathcal{A}}(b)$ be the random variable denoting the number of blocks mined by honest miners and adversary in a given time interval of length b respectively. As we assume that $\mathcal{A}$ withholds a block for at most t^* time before the honest miner accepts them, blocks in $A(s)$ are either mined by the adversary during $(t-s-t^*, t)$ or mined by honest nodes during $(t-s-\Delta, t)$. For any $\epsilon_0 > 0, \epsilon_1 > 0$, let $s_0 = \max\{\frac{\Delta}{\epsilon_0}, \frac{t^*}{\epsilon_1}\}$. Then $\forall\ s \geq s_0, s + \Delta < (1+\epsilon_0)s$ and $s + t^* < (1+\epsilon_1)s$. Hence,

$$A(s) \leq X_{\mathcal{H}}(s+\Delta) + X_{\mathcal{A}}(s+t^*) \tag{7}$$

$$\leq X_{\mathcal{H}}((1+\epsilon_0)s) + X_{\mathcal{A}}((1+\epsilon_1)s), \forall\ s \geq s_0 \tag{8}$$

Let $X(s)$ be a random variable denoting the number of blocks generated by a Poisson process within a time interval of size s with arrival rate $\bar{\lambda} = (1+\epsilon_0)\alpha + (1+\epsilon_1)\beta$. Since independent Poisson random variables are additive, we have the equality in distribution,

$$X(s) \stackrel{d}{=} X_{\mathcal{H}}((1+\epsilon_0)s) + X_{\mathcal{A}}((1+\epsilon_1)s) \tag{9}$$

Hence using equation 4, we have is

$$\Pr[Q(t) > b] \tag{10}$$

$$\leq \Pr\left[\bigcup_{s>0}\left\{A(s) - \frac{s}{\tau} > b\right\}\right] \qquad \text{(From equation 4)}$$

$$= \Pr\left[\bigcup_{s>s_0}\left\{A(s) - \frac{s}{\tau} > b\right\}\right] + \Pr\left[\bigcup_{s\leq s_0}\left\{A(s) - \frac{s}{\tau} > b\right\}\right]$$

$$\leq \Pr\left[\bigcup_{s>s_0}\left\{X(s) - \frac{s}{\tau} > b\right\}\right] + \Pr[A(s_0) > b] \tag{11}$$

$$\leq \Pr\left[\bigcup_{s>0}\left\{X(s) - \frac{s}{\tau} > b\right\}\right] + \Pr[A(s_0) > b] \tag{12}$$

The first term of equation 12 is the standard M/D/1 tail queue probability with arrival rate $\bar{\lambda}$, processing rate $1/\tau$ and hence its tail distribution probability decreases with increasing ζ. For any given b, t^*, ϵ, and Δ,

$$\Pr[A(s_0) > b] \leq 1 - \sum_{i=0}^{b-1} \frac{\bar{\lambda}^i e^{-\bar{\lambda}s_0}}{i!}. \tag{13}$$

□

C HIGH τ IN LEGACY POW BLOCKCHAINS

Recall from §3, on arrival of new block, a miner first validates the received block for τ units of time and then creates the next block for another τ units of time. Only after 2τ time units the miner starts PoW for the next block. Whereas when a miner mines the block himself it only spends τ creating the next block before starting PoW. Note that in case a miner receives the next block while validating the previous one, the time for which the miner needs to wait before it could start PoW is higher than 2τ. The exact waiting time depends on the exact time of arrival. We make a simplification and assume that all miners (including adversary $\mathcal{A}$) releases blocks only after τ units of time has passed from the broadcast of the previous block. We also assume that the network is fully synchronous.

Let $\Lambda = \{\lambda_j | j = 1, 2, \ldots, |N|\}$ be the block arrival rates due to a miner n_j during the periods they are performing PoW. Let $\tau_1$and τ where $\tau_1 = c\tau$ for $0 \leq c \leq 1$ denote the time required to validate a full block by miner n_1 and other miners n_j for $j > 1$ respectively. Note that $c < 1$ implies that n_1

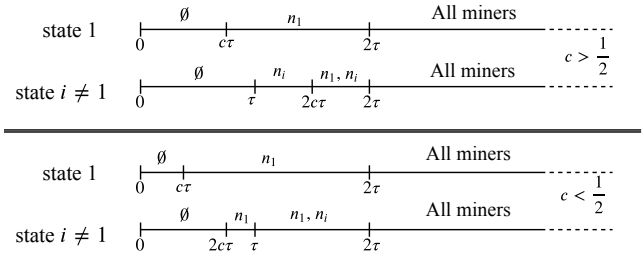


Fig. 14. Miners who are active during each time interval from the instant when last block gets generated in a situation where all miners are honest.

can validate (and also create) blocks faster than others. For example, $c = 1/2$ implies n_1 takes half the time than others take to validate or create blocks. Let $U = \{1, 2, \cdots, |N|\}$ be the states of the MC where state u represents that the miner of the latest block is n_u. State transition happens on arrival of every block and $p_{u,v}$ denote the transition probability for the transition from state u to state v. Note that on every state transition to state v, n_u mines a single block.

All honest miners. Since all miners in Ethereum do not always start PoW simultaneously, transition probabilities in the MC must consider only active miners. For example, for $c > 1/2$, in state 1, no miner performs PoW for the first $c\tau$ time units. Between time $(c\tau, 2\tau]$ only n_1 does PoW while other miners were busy validating and creating blocks. On the contrary, at any other state v, n_v starts PoW at time τ, n_1 starts at $2c\tau$ and the remaining start at time 2τ. This is depicted in the top diagram of Figure 14. Similarly, the bottom diagram in the same Figure illustrates the miners that perform PoW for at different intervals starting from different state. In this paper we only derive transaction probabilities of the MC for $c > 1/2$ as one can easily derive for $c \leq 1/2$ using similar approach.

Lemma C.1. *Let X, Y be two independent random variable with exponential distribution with arrival rate λ_x, λ_y respectively. Then the probability of the event $A = (X \leq \tau \wedge X \leq Y)$ is denoted using $N(\lambda_x, \lambda_y, \tau)$ and is equal to:*

$$\begin{aligned} Pr[A] &= N(\lambda_x, \lambda_y, \tau) \\ &= \frac{\lambda_x}{\lambda_x + \lambda_y}\left(1 - e^{-(\lambda_x+\lambda_y)\tau}\right) \end{aligned} \tag{14}$$

Proof. For any arbitrary time t where $0 < t \leq \tau$, $\Pr[t - dt \leq X \leq t] = f_X(t)dt = \lambda_x e^{-\lambda_x t}dt$ and $\Pr[Y > t] = 1 - \lambda_y e^{-\lambda_y t}$. Thus to get the closed form expression for $\Pr[A]$, we compute,

$$\begin{aligned} \Pr[A] &= \int_0^\tau f_X(t)\Pr[Y > t]dt \\ &= \int_0^\tau \lambda_x e^{-\lambda_x t} e^{-\lambda_y t} dt \end{aligned} \tag{15}$$

Solving the above produces the desired result. □

Theorem C.2. *For a particular $c > 1/2$, τ and $\Lambda = \{\lambda_j | j = 1, 2, \ldots, |N|\}$, with every miner honestly following the protocol, the state transition probabilities are given as:*

$$p_{1,1} = 1 - M_0(\lambda_1, 2\tau - c\tau) + p_1 M_0(\lambda_1, 2\tau - c\tau) \tag{16}$$

$$p_{u,1} = M_0(\lambda_u, 2c\tau - \tau)\,[N(\lambda_1, \lambda_u, 2\tau - c\tau) + M_0(\lambda_1 + \lambda_u, 2\tau - c\tau)p_1] \tag{17}$$

$$p_{1,v} = p_v M_0(\lambda_1, 2\tau - c\tau) \tag{18}$$

$$p_{u,u} = 1 - M_0(\lambda_1, 2c\tau - \tau) + M_0(\lambda_1, 2c\tau - \tau)\,[N(\lambda_u, \lambda_1, 2\tau - c\tau) + M_0(\lambda_1 + \lambda_u, 2\tau - c\tau)p_u] \tag{19}$$

$$p_{u,v} = M_0(\lambda_1, 2c\tau - \tau)M_0(\lambda_1 + \lambda_u, 2\tau - 2c\tau)p_v \tag{20}$$

with $N(\lambda_u, \lambda_v, t)$ is as given in Lemma C.1 and $M_0(\lambda_x, t)$ is the probability of 0 arrival in a Poisson process in a time interval t with arrival rate λ_x. Hence $M_0(\lambda_x, \tau) = e^{-\lambda_x \tau}$.

Proof. Transition of state 1 to 1 can happen either if n_1 mines the next block during $2\tau - c\tau$ interval or if n_1 mines the block after time 2τ. The former happens with a probability $1 - M_0(\lambda_1, 2\tau - c\tau)$. The latter happens with a probability p_1 conditioned on that the former event did not happen. Hence the probability of the latter is $p_1 M_0(\lambda_1, 2\tau - c\tau)$. Also since these two events are mutually exclusive, $p_{1,1}$ is sum of the probability of the events. Similarly, starting with state 1, any other miner n_v will mine the next block only if n_1 does not mine the block during an interval of length $2\tau - c\tau$ starting at $c\tau$. Also since all miners will be mining after time 2τ if no block was mined before that, the probability that the next winner would be n_v is p_v. Hence the transition probability $p_{1,v}$ equal to $p_v M_0(\lambda_1, 2\tau - c\tau)$.

Alternatively, starting from a state u with $u \neq 1$, n_u will mine the next block during time interval $(\tau, 2c\tau]$ with probability $1 - M_0(\lambda_u, 2c\tau - \tau)$. Otherwise n_u can mine the block during $(2c\tau, 2\tau]$. But as both n_1 and n_u will be mining during $(2c\tau, 2\tau]$, the probability of n_u mining the block before n_1 is equal to $N(\lambda_u, \lambda_1, 2\tau - 2c\tau)$. Lastly if n_u mine the block in neither of these interval, n_u will mine the next block with probability $p_u = \lambda_u/\lambda$. Combining the above will give the transition probability $p_{u,u}$. The transition probability $p_{u,1}$ can be derived similarly.

Lastly, state transition from a state u to v with $u \neq v \neq 1$ can happen if neither n_1 nor n_u mines a block prior time 2τ. Hence the transition probability $p_{u,v}$ is equal to $M_0(\lambda_1, 2c\tau - \tau)M_0(\lambda_1 + \lambda_u, 2\tau - 2c\tau)p_v$ □

Using the above state transition probabilities and mining power from Table 1 we numerically compute the stationary probabilities of the Markov chain with all miners being honest. Figure 3 illustrates our results for different c with varying τ.

Higher τ in the presence of an Adversary. Let node n_1 with arrival rate λ_1 be controlled by an adversary $\mathcal{A}$. We consider two different behaviors of the adversarial node n_1. First, n_1 validates the received blocks as per the protocol but instantly creates a block by putting transactions whose execution results are already known to n_1. In second n_1 skips validation of the received block as well and instantly starts PoW on top a new full block. The former attack is very practical as any miner can do that without any additional computational resources. The later damages fairness more severely but requires n_1 to produce final state due to transactions in the received block without executing them. An adversary can launch the later attack if it can download the modified state due to previous block from the creator of of the previous block. Figure 15 illustrates which miners do PoW at different time intervals starting from the instant of successful PoW on the previous block. The diagram at the top is when n_1 skips only creation and in the diagram at the bottom is when n_1

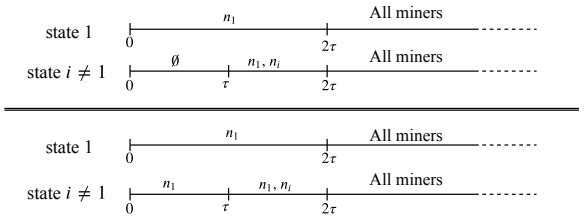


Fig. 15. Miners who are active during each time interval from the instant when last block gets generated. The diagram in the top corresponds to a adversarial node n_1, who only skips creation of blocks and the diagram at the bottom is in when n_1 skips both validation of received block and creation of new ones.

skips both validation and creation. Here, we will only derive the transition probabilities for the latter, and the transition probabilities for the former can be derived similarly.

Theorem C.3. *Given τ and $\Lambda = \{\lambda_j | j = 1, 2, \ldots, |N|\}$, with λ_1 as arrival rate of the adversarial node n_1, the transition probabilities for the Markov chain when n_1 skips validation of received blocks and the creation of new ones are:*

$$p_{1,1} = 1 - M_0(\lambda_1, 2\tau) + p_1 M_0(\lambda_1, 2\tau) \tag{21}$$

$$p_{1,v} = p_v M_0(\lambda_1, 2\tau) \tag{22}$$

$$p_{u,1} = 1 - M_0(\lambda_1, \tau) + M_0(\lambda_1, \tau)[N(\lambda_1, \lambda_u, \tau) + M_0(\lambda_1 + \lambda_u, \tau)p_1] \tag{23}$$

$$p_{u,u} = M_0(\lambda_1, \tau)[N(\lambda_u, \lambda_1, \tau) + M_0(\lambda_1 + \lambda_u, \tau)p_u] \tag{24}$$

$$p_{u,v} = M_0(\lambda_1, \tau) M_0(\lambda_1 + \lambda_u, \tau) p_v \tag{25}$$

Proof. When n_1 mines the last block, all other nodes will start PoW for the next block only after 2τ time units whereas n_1 will do PoW for the entire 2τ interval. Hence transition from state 1 to 1 will happen if either n_1 mines the block in the first 2τ time interval or n_1 mine the after 2τ time units. The former happens with a probability $1 - M_0(\lambda_1, 2\tau)$ and the later can happen with probability p_1 conditioned on the former not happening. Hence $p_{1,1}$ is $1 - M_0(\lambda_1, 2\tau) + p_1 M_0(\lambda_1, 2\tau)$. Similarly, the transition from state 1 to another state can only happen if n_1 did not mine during the first 2τ time units. Also since all nodes will mining after 2τ time units the transition probability $p_{1,v}$ is equal to $p_v M_0(\lambda_1, 2\tau)$.

When a node $n_u, u \neq 1$ mines the last block, n_1 instantly starts PoW for the next block. Hence for the first τ units of time only n_1 will be mining as even n_u will be busy creating the next block. During time $(\tau, 2\tau]$ both n_1 and n_u be mining and after 2τ the rest of the miners will start PoW for the next block. Thus n_1 can mine the next block either during the first τ or in the interval $(\tau, 2\tau]$ or after 2τ. The first can happen with a probability $1 - M_0(\lambda_1, 2\tau)$, the second with probability $N(\lambda_1, \lambda_a, \tau)$ conditioned on that the first did not occur, and lastly n_1 will mine a block after time 2τ with probability p_1 in case no block was mined prior to 2τ. Combining the above will give us the transition probability $p_{u,1}$.

Similarly transition from state u to itself happens when n_u mines the next block either during time interval $(\tau, 2\tau]$ or after time 2τ. The former happens with probability $N(\lambda_u, \lambda_1, \tau)$ conditioned on the event that n_1 did not mine the next block during first τ time units and the later with probability p_u conditioned on neither n_u nor n_1 mining a block before 2τ. Finally, transition to state to a state $v, v \neq u \neq 1$, will only happen with if neither n_u nor n_1 mine the next block before 2τ. Hence the transition probability $p_{u,v}$ is equal to $M_0(\lambda_1, \tau) M_0(\lambda_1 + \lambda_u, \tau) p_v$. □

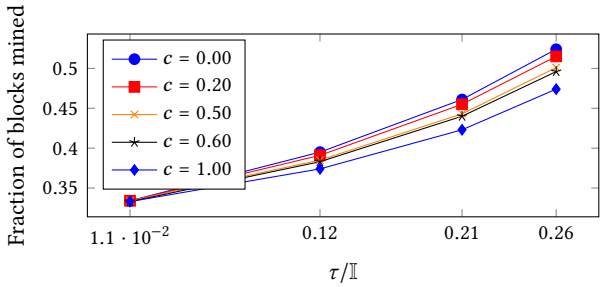


Fig. 16. Fraction of blocks mined by a adversary who validates a received block in time $\tau_1 = c\tau$ units of time, instantly creates block by putting transactions whose execution results are already known to the adversary, and controls approximately 0.33 fraction of the mining power.

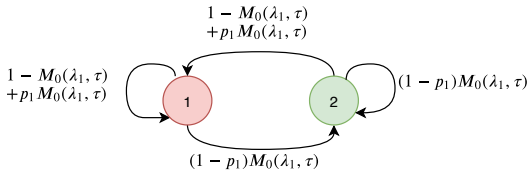


Fig. 17. Reduced Markov chain to analyze Skipping adversary. A general Markov chain for the skipping adversary can be reduced to the Markov chain above for large n where each adversary controls a f fraction of compute power and honest mining power is distributed among the remaining $n-1$ honest power such that the maximum amount of compute power a single miner controls extremely small.

Figure 16 illustrates numerically computed fraction of blocks mined by node n_1 which can validate blocks faster (by factor $1/c$) and skips creation of blocks. Note that $c = 0.0$ in the figure corresponds to the case where n_1 skips both validation of received blocks and creation of next ones.

Closed-form probabilities. In a system with n miners, one needs to solve a system of n linear equations to get closed-form equations for the stationary probabilities. We do not solve these as a part of this paper. But we compute closed-form stationary distribution for a particular case is given below.

Let there be K nodes in the network with an equal mining power of each node. Among these K nodes an adversary $\mathcal{A}$ controls a f fraction of the node and all adversarial nodes skips both validation and creation of blocks.

Consider an honest node which has just mined a block and just finished creating a new block. We assume that K is large enough to ensure that the probability of this honest miner successfully mining in the next τ units is approximately zero. In other words, we assume that the probability of all honest nodes together mining in this interval is zero and so only $\mathcal{A}$ mines in this interval. Under this assumption the Markov chain discussed so far can be reduced to a MC with only two states as depicted in Figure 17.

State 1 (resp. state 2) represents that the last state was mined by a adversarial (resp. honest) node. The transition probabilities are given as:

$$p_{0,0} = p_{1,0} = 1 - M_0(f\lambda, 2\tau) + fM_0(f\lambda, 2\tau) \tag{26}$$

$$p_{0,1} = p_{1,1} = (1-f)M_0(f\lambda, 2\tau) \tag{27}$$

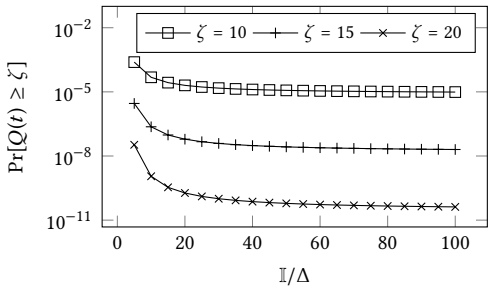


Fig. 18. Upper bound on the probability of queue at a honest miner crossing the threshold in the absence of an adversary for $\tau/\mathbb{I} = 0.5$

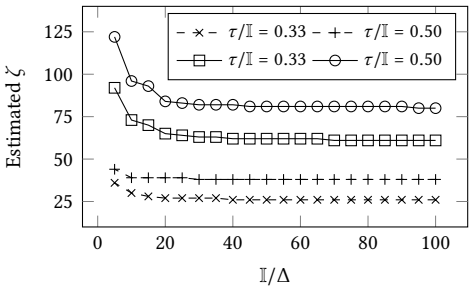


Fig. 19. Required ζ to upper bound $\Pr[Q(t) \geq \zeta]$ with 0.01. Dashed lines correspond to $f_{\max} = 0.25$ and solid lines are for $f_{\max} = 0.33$.

Let $\mathbf{P}_r$ be the transition probability matrix of this Markov chain and let $\Psi = (\psi_1, \psi_2)$ be the stationary probabilities of the states 1, 2 respectively. Then we solve, $\Psi\mathbf{P}_r = \Psi$ to get the stationary probabilities and these values are:

$$\psi_1 = 1 - M_0(f\lambda, 2\tau) + fM_0(f\lambda, 2\tau), \tag{28}$$

$$\psi_2 = 1 - \psi_1 \tag{29}$$

D QUEUE OVERFLOW IN HONEST EXECUTION.

In this section we will evaluate the performance of Tuxedo when all parties are honest as we believe this will be the most likely case.

For any given ζ, λ, Δ, and τ we compute the bounds using equation 12 by putting $\beta = 0$ and $t^* = 0$. This corresponds to a execution of Tuxedo in the absence of an adversary. Figure 18 illustrates the upper bound on the fraction of time queue at a honest miner will have more than ζ blocks for different values of ζ. All plots are for $\tau/\mathbb{I} = 0.5$. Note that in the absence of $\mathcal{A}$, for $\mathbb{I}/\Delta = 10$, with ζ as low as 20 queue, less than one in a billion honest blocks will hit a queue larger than ζ.

E CONCRETE CHOICE OF ζ.

For any given λ, $f_{\max}$(percentage of adversarial nodes), Δ, and τ, we evaluate ζ such that $\Pr[Q(t) \geq \zeta] < 0.01$. For all our evaluation we have used $\eta = 0.001$ in equation 5. Figure 19 illustrates our results for different values of $\tau/\mathbb{I}$ and $\mathbb{I}/\Delta$. For each $\tau/\mathbb{I}$ and $\mathbb{I}/\Delta$, we pick s_0 that minimizes ζ. For example, with 25% adversary and allowable processing time equal to half of average interarrival time, i.e. $\tau/\mathbb{I} = 0.33$, we get $\zeta = 26$ for $\mathbb{I}/\Delta = 30$.

Received August 2021; revised October 2021; accepted November 2021