Supervisory Control of Blockchain Networks

Kiam Tian Seow, *Senior Member, IEEE*

***Abstract*—Blockchain is an open distributed ledger technology that enables ledger-maintainers on a network to collaboratively synchronize and update their own distributed copies of a single global ledger, with the goal of keeping the ledger copies consistent. This paper presents a theoretical control-model formulation of the founding Satoshi Nakamoto blockchain, aimed at enhancing our operational understanding and development of blockchain systems. The control model is generic of every honest ledger-maintainer's local operations on a blockchain network. The presented research is a logical systematization of operational knowledge that is understandable and explainable for blockchain system engineering and research. Using a software tool supporting a supervisory control theory applied in the formulation, the control model is synthesized and logically validated.**

***Index Terms*—Discrete-event systems, distributed systems and consensus, formal languages and automata, operational modeling, Satoshi Nakamoto blockchain, supervisory control.**

I. Introduction

Consensus to achieve or maintain some form of data consistency[1] is a foundational idea of distributed systems. The problem of consensus search or determination and its local activation under operational control by consistency maintenance software agents, or simply maintainers, has been extensively studied and formalized in the distributed systems literature for the past three decades. However, the literature till then was almost all anchored in classical network settings - settings which are private or permissioned. As noted in [1], a departure from classical settings originated with the design of public peer-to-peer systems such as *Napster* and *Gnutella* for file sharing. This paper studies an important, but different, class of public peer-to-peer systems that record transactions between peers onto virtually a single global ledger - the shared system of transaction records. Specifically, it develops and examines a formal systematization of the operational control around a different form of consensus determination in a distributed ledger system called a blockchain [2], a disruptive peer-to-peer system that has been touted to 'change the world' [3] by revolutionizing contracts and human interactions on an Internet scale. In a public or permissionless network setting that is unlike any before it, the ledger-distributed and operation-decentralized system is realized with anonymous ledger-maintainers, whose participation to synchronize and update their own distributed copies of a single global blockchain ledger, to keep them consistent, is not known *a priori*. Maintainers, each as a node of the network, can dynamically join and leave the system network without seeking permission from a centralized or distributed authority. Their consensus on whose newly assembled block of transactions to add next to update and maintain consistency among their ledger copies is determined with honest maintainers forming some majority. Fundamentally new in thought as expounded in the founding work [2] is the workable idea of honest majority - a notion defined by a decisive percentage of the total computational resources held by honest maintainers, and not by their decisive population size. This honest power-majority also has to be made effective, by way of auto-setting mathematical races for winning (and thus determining) consensus to be computationally harder for dishonest maintainers (in the power-minority) to consistently finish sooner than honest ones. In practice, each maintainer node runs on purpose-built machines packed with custom chips [3] that contribute unbeknown to their side of the collective computing power, honest or otherwise. In what follows, to underline the importance of blockchain research, the rest of this introduction section first explains what a founding blockchain does and how it works, as well as the ledger features it possesses by design and the problems these features could solve that together make blockchain potentially so disruptive. It then points out an ultimate goal of blockchain research, before presenting the research motivation and the formal approach of this paper - that of systematizing blockchain operational control.

The founding blockchain, in the style of Satoshi Nakamoto, is the open foundational ledger technology underlying Bitcoin [2], a digital currency created in 2009, and is adopted by almost all other contemporary digital currencies (e.g., Ethereum, Litecoin, and Dogecoin) and related services [4]. Depending on context, 'blockchain' refers to either the distributed system or a distributed copy of the single global ledger locally owned and updatable by a maintainer of the system. A peer-to-peer network, blockchain operates on the IP protocol on the Internet, requiring neither pre-established trust between peers nor a 'trusted' third party as middleman whose service integrity cannot be scientifically guaranteed. As claimed, it allows peers as transaction makers on a network to securely transact ownership transfers between themselves, with distributed ledger-maintainers continually validating and serially ordering new transactions when assembling new transaction blocks, and updating their own ledger copies each time with a common new block. Appending block by block, to regularly extend and form the longest chain of blocks which is the only chain mandated as correct for every maintainer in their ledger copy, the goal of such update synchronization is to maintain consistency among the longest chains locally formed in their ledger copies. This is carried out by an innovative consensus-based protocol designed to work with no middleman involvement, such that the distributively shared ledger cannot be tampered with without detection. Under this operationally decentralized protocol, it is necessary for honest (computer)

K.T. Seow is with the Robot Intelligence Technology Laboratory, School of Electrical Engineering, KAIST, Daejeon 305-701, South Korea. E-mail: ktseow@rit.kaist.ac.kr

[1]The strictest of which is the status that all nodes in the network have the latest data history at the same time.

maintainers to be in the effective power-majority; and the consensus, on whose assembled transaction block to append next in every maintainer's distributed ledger copy, is decided by who succeeds first in completing a computational race. This race is costly and entails solving a mathematical puzzle based on a cryptographic hash function [5] to obtain a solution as some verifiable proof of work (PoW). PoW is presented as evidence of spent resources by which the human owner of the successful maintainer may be subsequently rewarded financially for its block contribution.

To utilize it as a critical infrastructure, a blockchain system needs to be secure against dishonest ledger-maintainers adding consecutive blocks containing fraudulent transactions in a new branch or 'fork', 'lengthening' it to successfully form the longest ledger chain in the process and thus corrupting the ledger. By design, the application promise of blockchain technology in delivering optimistic use-value is facilitated by the following three features built into the ledger:

1) Authenticability - whether or not a thing, including a person, is what it says it is, is publicly checkable.
2) Transparency - validated transaction records in the blockchain are publicly viewable.
3) Auditability - validated transaction records are independently verifiable.

However, this promise makes sense provided hash-chaining the timestamped blocks that validated transactions are serially assembled into, as done under the blockchain protocol to successively append blocks in a chain, can secure or tamper-proof the blocks (against ledger corruption) once the blocks are confirmed in the ledger. Making good on the promise would open up an exciting world of potentially disruptive blockchain applications for business, government and society where trust, transactions and contracts[2] are their underpinnings, to deal with the inherent uncertainty in identity management, asset tracking and reneging on deals, all without entrusted third parties or central authorities. These parties or authorities include governments, banks, accountants, notaries, and paper monies - potentially central points of failure that have happened before, but which businesses continue to rely on heavily to mitigate such uncertainty in real-world environments.

To elaborate in more detail [6], *identity management* faces the uncertainty problem of not knowing who or what things we (the buyer or seller, or more generally the ownership transferor or transferee of assets[3]) are dealing with; *asset tracking* faces the problem of us not having the visibility of transaction executions that move assets around - for example, how did a product get to us, and as a result, is the product we ordered the same as the one we received?; and *reneging on deals* faces the problem of us not having recourse if things go wrong - for example, can we get our money back if we do not receive our ordered goods or receive them on time? How well blockchain technology can address these mutually dependent problems of transaction uncertainty depends on how securely the following can be realized in the blockchain network to, respectively, address them:

1) Providing a cryptographic proof of identity for authentication purposes, as a digital signature for checking by individual network users based on public key cryptography, whereby each user is assigned with a private key and the matching public key is made known to all other users in the network.[4]
2) Generating a 'shared reality' (or consensus) of transaction records for transparent monitoring and verification by individual network users.
3) Allowing application developers to write codes to bind contracts of deals as transactions between network users for auditable self-execution.

Distilled from the Bitcoin application that it is motivated by as a no-middleman solution to tackle the problems of transaction uncertainty between users, blockchain builds what are hereby called Decentrally Affirmed, Ownership-Transfer Transactions Networks (DAO-T2Nets), to highlight blockchain's core operational purpose. An ultimate research goal is then to make DAO-T2Nets scientifically secure so as to be reliable or entrustable in mitigating the uncertainty problems in business and social transactions. To achieve this research goal, there is a range of challenging security and privacy issues in blockchain technology that needs to be comprehensively uncovered and adequately addressed in the presence of possible adversaries - the dishonest ledger-maintainers. A known number of these issues have been surveyed and discussed in the literature [7], [8], [9]. These issues span across and blend various research disciplines including cryptography, distributed systems and consensus, and the economics of incentives. A DAO-T2Net system may be confounded by these issues at different operational stages - from transaction creation to block addition in the blockchain. As a coherent guide to blockchain research including and beyond the founding version [2], a useful six-layer reference framework is recently proposed in the literature [10].

Focusing on foundation, this research posits that before we could, as a research community, more holistically understand and effectively address security and privacy issues to securely realize the application promise of blockchain technology, we need to qualitatively model local blockchain operations at an honest ledger-maintainer or miner node of the network in a systematic way. As perhaps the first efforts in this direction, this paper focuses on modeling the Satoshi Nakamoto style of blockchain operation [2] for DAO-T2Net systems. The modeling is of how an honest maintainer operates generically, regardless of whether or not dishonest maintainers are present in the same network environment. The purpose is to distil the basic operational characteristics as well as related update-synchronization and collaboration concepts in an implementation-independent fashion, to foster a common understanding of how honest maintainers collaboratively operate the blockchain in a possibly adversarial environment.

[2]For Bitcoin, a contract specifies conditions to be fulfilled for executing transactions.

[3]An asset refers to anything of value, including a vote, a patentable idea, a digital right, etc., besides money.

[4]In blockchain identity authentication, the only check performed is whether or not a transaction was signed by the correct private key. Anyone who has access to the private key is assumed the transaction initiator and sender, and the exact identity of the initiator is deemed irrelevant.

To do so, a supervisory control theory of discrete-event systems (DES's) [11], [12], [13] is applied. The term 'discrete event', or simply 'event', defines a qualitative change signaling what distinctly changed (and not how much of it changed) that evolves a system from one to another (possibly unchanged) set of distinct conditions called a state. An event can be a specific action taken (e.g., button pressed), a spontaneous occurrence dictated by nature (e.g., sensor failed), or an abrupt fulfillment of some defined condition (e.g., buffer filled). A DES in control engineering is an event-driven system whose state evolution over time starting from an initial state depends entirely on the asynchronous occurrence of events. Originally founded on a mathematically rigorous foundation of formal languages and finite automata [14], the control theory helps to conceptualize a problem neatly into a system part and a system requirement specifications part, and is supported by design software tools to automatically synthesize and validate an appropriate solution, which is the control part supervising the system part to meet the specifications.

In outlining the approach, this research identifies and models the local operational tasks of an honest blockchain ledger-maintainer as generic interleaving discrete-event processes constituting the system part. The research then shows that these processes are behaviorally controlled by a supervisory process - the maintainer operational model constituting the control part in conjunction with the system part, that guarantees proper ledger-update synchronization as specified. All modeled in finite automata, as we shall see, the solution model is understandable and explainable in terms of surprisingly simple constituent parts, namely the modeled processes of the blockchain system and the ledger-update synchronization specifications.

The rest of this paper is organized as follows. Section II reviews the relevant DES terminology and results of supervisory control in formal languages and finite automata. Section III provides an operational overview of the Satoshi Nakamoto blockchain. Basing and expanding on the overview description, Section IV proposes DES models for the system and the specification parts for designing an honest blockchain ledger-maintainer. Section V then presents the design synthesis of the ledger-maintainer as a control model based on the proposed models, and a validation of the control model. Section VI discusses the potential impact of the control model in the context of related work. Section VII concludes this paper.

II. Supervisory Control Theory

This section provides the relevant background on supervisory control of discrete-event systems (DES's) [11], [15], [16], [17] founded on formal languages and finite automata [14]. The material is taken primarily from the monograph [12].

A. DES Modeling in Formal languages & Finite Automata

Let Σ be a finite set of symbols representing events, and Σ^* be the set of strings over Σ, including the empty string ε (a sequence with no events), where a string is a finite sequence of events. Given a string $s \in \Sigma^*$, a string s' is a prefix of s, denoted by $s' \leq s$, if $(\exists t \in \Sigma^*)s't = s$.

Defined over Σ, a formal language K is a subset of Σ^*. The prefix closure of K, denoted by $\overline{K}$, is $\overline{K} = \{s' \mid (\exists s \in K)s' \leq s\}$, the language of all prefixes of strings of K. Note that $K \subseteq \overline{K}$, and $K \neq \emptyset$ provided $\varepsilon \in \overline{K}$. The language K is said to be prefix-closed if $K = \overline{K}$. For $K_1, K_2 \subseteq \Sigma^*$, K_1 is said to be a sublanguage of K_2 if $K_1 \subseteq K_2$. K_1 and K_2 are said to be nonconflicting [12] if $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$.

A language is said to be regular provided it can be generated by a finite (-state) automaton [14] or simply an automaton. An automaton G is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite state set, Σ is the finite event set, $\delta : \Sigma \times Q \to Q$ is the (partial and deterministic) transition function, q_0 is the initial state, and $Q_m \subseteq Q$ is the subset that contains the marked states. The transition function δ can be extended to Σ^* as follows: $\delta(\varepsilon, q) = q$, and $(\forall \sigma \in \Sigma)(\forall s \in \Sigma^*)\delta(s\sigma, q) = \delta(\sigma, \delta(s, q))$, which is defined if $q' = \delta(s, q)$ and $\delta(\sigma, q')$ are both defined.

The behavior of automaton G is described by two languages, namely the prefix-closed language $L(G) = \{s \in \Sigma^* \mid \delta(s, q_0) \text{ is defined}\}$ and the marked language $L_m(G) = \{s \in L(G) \mid \delta(s, q_0) \in Q_m\}$. By definition, $L_m(G) \subseteq L(G)$, and is the sublanguage modeling strings that have some specified purpose, such as the completion of a task.

A state $q \in Q$ is reachable (from the initial state q_0) if $(\exists s \in \Sigma^*)\delta(s, q_0) = q$, and coreachable if $(\exists s \in \Sigma^*)\delta(s, q) \in Q_m$. Automaton G is said to be reachable if all its states are reachable, coreachable if all its states are coreachable, by which $\overline{L_m(G)} = L(G)$, and trim if it is both reachable and coreachable.

An automaton G that is not trim can be trimmed to one computed as $Trim(G)$ [12], where $L(G) \supsetneq L(Trim(G))$ if G is not trim, but $L_m(G) = L_m(Trim(G))$, i.e., the marked language of G is preserved by $Trim$. Automaton $Trim(G)$ is said to correctly model $L_m(G)$. Therefore, for G to model a regular $K \subseteq \Sigma^*$ as $K = L_m(G)$, G is necessarily trim.

The projection function P that masks out or erases all the occurrences of every event in a specified subset $MASK \subseteq \Sigma$ from a string $s \in \Sigma^*$ is defined as follows: $P : \Sigma^* \to (\Sigma - MASK)^*$, where $P(\varepsilon) = \varepsilon$, and $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)$ $P(s\sigma) = P(s)\sigma$ if $\sigma \in \Sigma - MASK$, and $P(s)$ otherwise. This function P is called the natural projection of Σ^* onto $(\Sigma - MASK)^*$. It follows that the projection or abstraction of an automaton G, with events in $KEEP = (\Sigma - MASK)$ retained, is an automaton A computed as $Project(G, KEEP)$ [12], i.e., $A = Project(G, KEEP)$, such that $L(A) = \{P(s) \mid s \in L(G)\}$ and $L_m(A) = \{P(s) \mid s \in L_m(G)\}$.

Graphically, an automaton G is an edge-labeled directed graph represented as follows: A graphical node denotes an automaton state. A σ-labeled edge, directed from a node denoting a state q to a node denoting a state q', represents $\delta(\sigma, q) = q'$, the transition of event σ from q to q'. A node with an entering arrow denotes the initial state q_0, and a node that is darkened denotes a marked state.

An automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ is often modularly formed by n component automata $G_1, G_2, \cdots, G_n$, $n \geq 2$, with $G_i = (Q_i, \Sigma_i, \delta_i, q_{i,0}, Q_{i,m})$ $(1 \leq i \leq n)$, whose interactions among them is modeled on the synchronous operator $\|$ [18]; and is denoted by $G = G_1 \parallel G_2 \parallel \cdots \parallel G_n$, called the synchronous product. For $n = 2$, the synchronous product $G = G_1 \parallel G_2$ models G_1 and G_2 interacting by interleaving events generated by G_1 and G_2, with synchronization of shared events in $\Sigma_1 \cap \Sigma_2$, and is constructed as follows: $Q = Q_1 \times Q_2$, $Q_m = Q_{1,m} \times Q_{2,m}$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $q_0 = (q_{1,0}, q_{2,0})$ and $\delta(\sigma, (q_1, q_2))$ is defined by

$$\begin{cases} (\delta_1(\sigma, q_1), \delta_2(\sigma, q_2)), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \text{ \& both} \\ & \delta_1(\sigma, q_1) \text{ \& } \delta_2(\sigma, q_2) \text{ are defined} \\ (\delta_1(\sigma, q_1), q_2), & \text{if } \delta_1(\sigma, q_1) \text{ is defined \& } \sigma \notin \Sigma_2 \\ (q_1, \delta_2(\sigma, q_2)), & \text{if } \delta_2(\sigma, q_2) \text{ is defined \& } \sigma \notin \Sigma_1 \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

By the associativity of $\|$ [18], the modular automaton G for $n > 2$ can be recursively constructed as defined above.

If $\Sigma_1 = \Sigma_2$, the synchronous product $G = G_1 \parallel G_2$ reduces to the (reachable) Cartesian product [18], modeled on the Cartesian operator $\sqcap$ and denoted by $G = G_1 \sqcap G_2$, for which $L(G) = L(G_1) \cap L(G_2)$ and $L_m(G) = L_m(G_1) \cap L_m(G_2)$.

Finally, note that $G_1 = G_2$, i.e., G_1 and G_2 are equivalent, provided $L(G_1) = L(G_2)$ and $L_m(G_1) = L_m(G_2)$.

B. Basic Control Problem, Solution Synthesis & Support

Let a reachable automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ model a DES[5], with the event set Σ partitioned into the controllable event set Σ_c and the uncontrollable event set Σ_u. By definition, a controllable event can be prevented from occurring, while an uncontrollable event cannot be. A specification language $K \subseteq \Sigma^*$ is said to be controllable with respect to DES G if $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$ [11]. Intuitively, that K is controllable (with respect to G) means that following an arbitrary string $s \in \overline{K} \cap L(G)$, the DES G does not slip out (of $\overline{K} \cap L(G)$, and hence $\overline{K}$) on an uncontrollable event.

Let an arbitrary reachable automaton S be a supervisor for a DES G, with S and G sharing the same event set Σ. By $S \sqcap G$, we may think of $\sqcap$ as a control operator that abstracts away the communication of the event feedback from DES G and the control by supervisor S. Then a problem of fundamental interest is to find a nonblocking supervisor S that can control DES G to meet a given specification language $K \subseteq \Sigma^*$, by enabling and disabling events in Σ_c while always allowing events in Σ_u to occur in the DES. Now:

1) By a nonblocking supervisor S for DES G, it means

$$\overline{L_m(S \sqcap G)} = L(S) \cap L(G),$$

noting that $L_m(S \sqcap G) = L_m(S) \cap L_m(G)$. Intuitively, S nonblocking means every string generated under control can be extended to a common marked string between S and G, indicating no task in DES G identified by S is blocked from completion by the control action of S. It is always possible and the practice to find a nonblocking S (out of possible candidate automata) that is also coreachable (and hence trim) with less or no unnecessary states for exerting the same control actions, and is therefore more concise.

2) By S controlling G to meet K, it means

$$L(S \sqcap G) \cap K = L_m(S \sqcap G) \subseteq K \cap L_m(G).$$

Stating more generally the solvability of the fundamental problem proved in [11], there exists such a nonblocking and K-meeting supervisor S, for which $L_m(S \sqcap G) = K \cap L_m(G)$, if and only if K is controllable and $\overline{K \cap L_m(G)} = \overline{K} \cap L(G)$.

A language $K \subseteq \Sigma^*$ may not be controllable. However, the supremal (or largest) controllable marked sublanguage of the DES G that lies within K exists [15]. This sublanguage can be generated by a trim automaton computed as $Supcon(G, K)$[6] [12], [15] for which an arbitrary S such that $S \sqcap G = Supcon(G, K)$ is a solution supervisor, and $\underline{\text{is exactly}}$ $K \cap L_m(G)$ provided K is controllable and $\overline{K \cap L_m(G)} = \overline{K} \cap L(G)$. The algorithmic procedure [15] '*gets the solution right*', in that it constructs the solution automaton $Supcon(G, K)$ that meets K, since $L(Supcon(G, K)) = \overline{L_m(Supcon(G, K))}$ - meaning DES G under the control of a nonblocking solution S will be kept within $\overline{L_m(Supcon(G, K))}$, and $L_m(Supcon(G, K)) \subseteq K \cap L_m(G)$ - 'target' of the specification K of interest. We may select $S = Supcon(G, K)$ as the nonblocking supervisor (for DES G to meet K).

The automaton $Supcon(G, K)$ represents the 'full' solution because it has 'embedded' in it all the *a priori* transitional constraints of DES G. As a result, it can be a lot larger in state size than is necessary to achieve the same control actions. A state reduced and trim automaton may be obtained as $Supreduce(G, A)$ [16] for S, where $A = Supcon(G, K)$. Automaton $Supreduce(A, G)$ has the *a priori* constraints of DES G relaxed as much as possible. Often greatly state reduced, $S = Supreduce(G, A)$, together with DES G in (synchronous) modular form $G_1 \parallel G_2 \parallel \cdots \parallel G_n$ $(n \geq 2)$, contributes to presenting $S \sqcap G$ as a more understandable solution than $Supcon(G, K)$ is.

Whether reduced or otherwise, because the supervisor S in $S \sqcap G = Supcon(G, K)$ generates the largest controllable sublanguage of K with respect to DES G, it is said to be optimal or maximally permissive (with respect to G under K conformance).

As a specification, language $K \subseteq \Sigma^*$ is often practically expressed in a (conjunctive) modular form $K_1 \cap K_2 \cap \cdots \cap K_p$ $(p \geq 2)$ - an intersection of two or more languages (with each over the same event set Σ). For $p = 2$, the following basic result presents a sufficient condition for the existence of a corresponding modular solution. It is a useful guide to developing a modular solution that is more insightful (if not more understandable) than the equivalent monolithic supervisor version, especially if its constituents, each computed using $Supreduce$, are simple and intuitive.

[5]The design practice, however, is to begin with a DES model G that is trim or made trim.

[6]Note that the language of specification interest for control synthesis is $K \cap L_m(G)$, which is a regular language because automaton G is finite-state, and can thus be modeled by an automaton. In the procedural computation [12] of $Supcon(G, K)$, K can be practically specified as a regular language by an automaton (which is necessarily trim).

Theorem 1 (On Modularity of Supervision [12]): Consider $K_1, K_2 \subseteq \Sigma^*$, and $K = K_1 \cap K_2$ for a DES G (over event set Σ). Suppose $S \sqcap G = Supcon(G, K)$, $S_1 \sqcap G = Supcon(G, K_1)$, and $S_2 \sqcap G = Supcon(G, K_2)$, where S, S_1 and S_2 are (nonblocking) supervisors for DES G, each with the same event set Σ. Then

$$S \sqcap G = (S_1 \sqcap S_2) \sqcap G$$

if $L_m(S_1 \sqcap G)$ and $L_m(S_2 \sqcap G)$ are nonconflicting.

A software tool TCT [19] is available for design, synthesis and validation, in finite automata, of systems applying the control theory. The tool is a formal methods library of algorithmic procedures. Besides $Trim$, $Project$, $Supcon$, and $Supreduce$[7], the library includes $Nonconflict$, $Sync$, $Meet$, and $Condat$. $Nonconflict$ is for testing if two (regular) languages modeled by trim automata are nonconflicting; $Sync$ implements the synchronous operator $\|$; $Meet$ implements the Cartesian operator $\sqcap$; $Condat$ is for use in testing the controllability of the prefix-closed language of an automaton. As listed online [20], other control design software tools are also available. TCT is developed by the founding group whose basic control theory is reviewed in this background section. In this paper, TCT is used to construct and validate the logical design of the Satoshi Nakamoto blockchain system [2].

As briefly described in the introduction, anonymous ledger-maintainers on a DAO-T2Net continually synchronize and update their own distributed ledger copies, and maintaining the consistency among their ledger copies hinges on honest maintainers being able to consistently contribute the transaction block-updates. In the next section, a more detailed operational description of the blockchain is first provided. In the description, eight events are identified and placed in brackets. From the description, the key operational processes and collaborative ledger-update synchronization requirements, by which an arbitrary ledger-maintainer asynchronously operates, are then modeled as (discrete-event) automata in Section IV; the state activities and conditions associated with every event are detailed as required to complete the modeling for an honest maintainer.

III. Description of Blockchain Operations

The blockchain operates in a totally decentralized fashion to cryptographically validate and record peer-to-peer transactions in a distributed (public) ledger of a transactions network. In the most basic case, a transaction records an ownership transfer. In a DAO-T2Net or blockchain network are anonymous nodes denoting two types of participating members - users (i.e., transaction makers or creators) and blockchain ledger-maintainers, who may, as in an open distributed system, join and leave the network. As users transact, every corresponding transaction message or simply transaction[8] is also sent to inform each maintainer. Blockchain ledger-maintainers work individually as miners that, supposedly as honest maintainers would, continually take from their incoming transaction message queue (new_tx_rcd) and validate the transactions (tx_vdx_dn) for integrity, and then assemble them serially in a new block, within which a Merkle Tree [21] is then constructed. The construction of a Merkle tree is done with the validated and serially ordered transactions placed at its base, and connected to their corresponding hashes (used as transaction identifiers) placed in the same order at the level immediately above the base. These ordered hashes are then consecutively paired and hashed, with no identifier considered in more than one pair, to form hashes at the next higher level that each corresponding pair is graphically connected to in the tree. This is followed by similarly forming hashes of consecutive pairings of the resultant ordered hashes at each subsequent higher level, until the root hash of the tree is formed. When forming the ordered hashes at each level in the tree for subsequent pairing and hashing, excluding the root, the last hash is duplicated whenever there is an odd number of hashes. Being uniquely dependent on what validated transactions are in the block and in what order, the root hash furnishes a cryptographic proof of no-tampering. The miners, including possibly dishonest ones, upon finishing their own new block assembly (nxt_svblk_rdy), possibly at different times, then begin to race. The race is by way of each miner computationally solving a block-dependent mathematical puzzle [5] of some difficulty level. This difficulty level is calibrated once every N blocks added by ledger-maintainers in the network to their ledger copy; the calibration is done by some external process based primarily on the network hash rate, i.e., the time duration ledger-maintainers last took to add N blocks to their ledger copy. Through this race, the miners arrive at a consensus (cp_solved) by which the winner's block of validated and serially ordered transactions is affirmed as the next block to append and broadcast ($vblk\_broadcasted$, with appending of the self-assembled-and-validated block by the winner in its own ledger copy subsumed).

The mathematical puzzle [5] is set based on the previous block identifier (ID) and the (current) candidate block of validated and serially ordered transactions, and is solved by Bernoulli trial and error (random search). The puzzle is outlined as follows: Given a puzzle difficulty level p, find x such that

$$H(\text{previous block ID}, \text{candidate block}, x) \leq target(p),$$

where H is a cryptographic hash function and $target(p)$ is the puzzle-difficulty threshold generated and set accordingly, such that the probability of a guess x (called a nonce) satisfying the inequality, i.e., making the resultant hash equal to or below the given target, is p, also called mining or puzzle hardness, or PoW difficulty[9]. For Bitcoin [2], H is the SHA-256 hash function modeled as a random oracle - a completely unpredictable pseudorandom function, and that is why the only way to find a nonce x satisfying the inequality is by trial

[7]Note: Based on the original conception [16], $Supreduce$ has been implemented to find a state reduced (trim) S such that $S \sqcap G = Supcon(G, K)$ for a given K modeled and input as a trim automaton, along with stating if the S found is state minimal. The latest version of the TCT software includes a 'clean-up' option of finding one such that $S \parallel G = Supcon(G, K)$. This paper adheres to the original conception.

[8]Typically, a transaction consists of date and time of transaction, participating users, and assets for ownership transfer.

[9]Note that a lower p $(0 < p < 1)$ indicates a higher PoW difficulty realized by a smaller threshold value $target(p)$, and conversely.

TABLE I
STRUCTURE OF A TRANSACTION BLOCK [2].

	Block				
	Header			Body	
Block identifier: Hash code of the block	Previous block identifier	Consensus puzzle solution as 'proof of work' (PoW), a nonce that is a correct guess	Root hash of Merkle tree (of the validated transaction block)	Merkle tree of (interior) hashes of validated transactions, with root hash in header	Block of validated transactions (at base of Merkle tree)
	Timestamp: Approx. block creation time	Puzzle-difficulty threshold *target(p)*, where *p* indicates the level of PoW difficulty set for the block			Block size & number of transactions in block

and error, repeatedly incrementing x and seeing if the new hash value matches. To an honest maintainer, the previous block ID is a block hash code obtained by hashing the header of the block (see Table I depicting the block structure in tabulated form) at the top of currently the longest chain of blocks first formed in the maintainer's ledger copy. Note that, on the condition that honest maintainers form and preserve the power-majority, the calibration of the consensus-puzzle difficulty p is aimed at allowing only honest maintainers to consistently win the computational race. Winning the race is about a ledger-maintainer finding a puzzle solution x before any other maintainer does. This should occur in a reasonably short time duration (once about every 10 min on the average for Bitcoin, achieved by readjusting p accordingly, once every 2016 blocks). But the race winner is to emerge only after the puzzle computation time duration has exceeded the maximum network time delay, since it is by which time that the previous affirmed block broadcasted would have been received by every maintainer [1], [2]. In general, based on the network hash rate measured, the longer the network time delay and the more ledger-maintainers there currently are in the network, the higher it is that the PoW difficulty level determined may need to be adjusted for the sake of ledger consistency [1].

In what follows, all ledger-maintainers in the network are to append the winner's newly assembled block ($vblk\_chained$) to, normally, the longest chain[10] in their individual ledger copies. This is after they have individually received and verified the puzzle solution as PoW obtained in the race, also sent along together with the affirmed block by the consensus race winner for block validation (new_blk_rcvd). For each time validating and affirming its assembled transaction block as the next block to be appended in the distributed ledger, the computer ledger-maintainer that won the consensus race has its human owner's digital pocket or wallet subsequently deposited with a financial reward once the block is confirmed[11] in the ledger. In this manner, the distributed ledger-updating work proceeds asynchronously among maintainers upon their every individual start of processing their next block ($nxt\_blkp\_started$), with the goal that such constant update synchronization under an effective honest power-majority can keep the entire peer-to-peer network's transaction history consistent in each maintainer's ledger copy. Every ledger copy as a result is a cryptographically-sealed chain of transaction blocks that is chronologically-ordered as mutually agreed by PoW-based consensus.

As discussed in [2], a weaker notion referred to as T-consistency [1] between ledger copies is actually used for blockchain. It is defined as two chains, the longest in the respective current ledger copies of two arbitrary maintainers, each differing from the other in at most their last $(T-1)$ consecutive blocks, where $T \geq 1$ is a relatively small number.

IV. MODELING BLOCKCHAIN PROCESSES & REQUIREMENTS

The blockchain operational description and identified events are mapped onto the supervisory control framework in finite automata. The 'mapping' is localized to operations within an arbitrary ledger-maintainer node of a DAO-T2Net, with the node infrastructure conceptualized as depicted in Fig. 1. The purpose is to construct the ledger-maintainer model as a supervisor of a system resident in the network node. The logical design mapping uncovers the local system at the node as a modular DES of four discrete-event processes, along with two system requirement specifications prescribing how the maintainer should collaboratively synchronize the block-by-block transaction updates of its ledger copy (with the other maintainers' in the network). In a top-down fashion, the underlying details of the events, including what activities accompany their occurrence or execution, are also added to refine the logical system model into one pertaining to an honest maintainer's. Using the models of the system and specifications formulated in this section, the ledger-maintainer operational model is then synthesized as a nonblocking and specifications-meeting supervisor of the system, and shown to be valid in the next section.

A. The Preliminaries

Referring to Fig. 1, the self-block buffer at every ledger-maintainer node is used for holding validated transactions when assembling a transaction block, and has a capacity set to the size that defines an admissible block. The received-block buffer is used for holding new, validated blocks received from the network, and has a finite capacity of multiple blocks.

[10]The 'length' of a chain is counted in terms of the amount of (block) mining work put into the chain. Practically, it is measured by the sum total of the PoW difficulty for every block already added in the chain.

[11] A transaction block is said to be confirmed, once and only when it is adjudged to be stochastically infeasible to reverse or modify any of the transactions recorded therein, and that is when the block is stochastically assured of always remaining in any longest chain of the ledger. In other words, block confirmation usually occurs when the block is further back in the chain, since any transaction in a block further back is computationally harder or stochastically more infeasible to reverse by the only ponderable way - that of attempting to form the longest chain by creating and lengthening a branch or fork of transaction blocks to extend a chain's prefix that only just excludes the block.

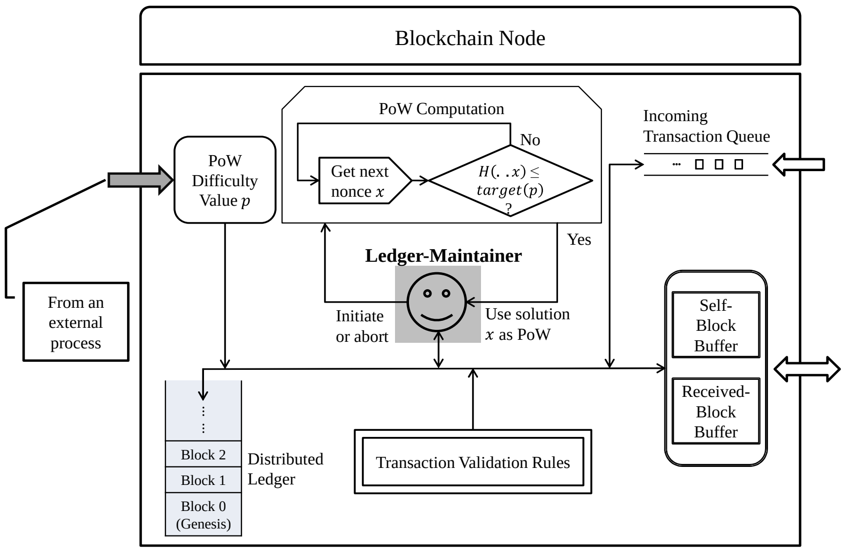


Fig. 1. The ledger-maintainer node infrastructure.

The same genesis block is created in the ledger copy at every maintainer node when a DAO-T2Net first comes alive. A maintainer node that subsequently joins or rejoins the network will inherit the latest ledger copy as proof of what has happened to begin with.

The start of processing a new block means that a computer ledger-maintainer has created and deposited in its self-block buffer the block's first transaction stating a financial reward, calculated based on some reward policy, that is to be deposited into the digital pocket of the maintainer's human owner if the block could be added in the ledger copy and subsequently confirmed. Transaction processors, including the reward payment processor, can publicly view and search any maintainer's ledger copy; and subject to their own execution policy, each processor can decide when to execute the transactions in a block added in the ledger copy that are under its jurisdiction, after the block has been confirmed.

B. The System Model

The local system at a ledger-maintainer node of a DAO-T2Net is modeled by the DES G,

$$G = G_1 \parallel G_2 \parallel G_3 \parallel G_4, \tag{1}$$

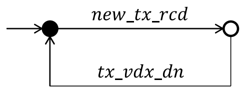
where each trim component process model G_i $(1 \leq i \leq 4)$, along with their defined events, is shown in Fig. 2.

Referring to Fig. 2, TX_VALIDATION G_1 performs transaction validation; BLK_INPUT G_2 takes a new validated transaction block as the next update input for the local ledger copy (i.e., the copy at the node), with the block either self-assembled or received from (another ledger-maintainer in) the network; and before starting the next local block-processing cycle, LEDGER_BLKUPDATE G_3 updates the ledger copy, each time with a block while CONSENSUS_FIND G_4 performs PoW computation in an attempt to win the next block-update consensus race, or signals that a block received from the network is ready for chaining to the ledger copy.

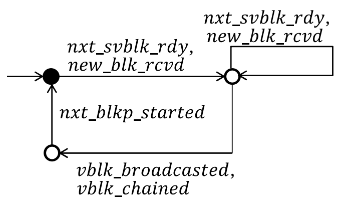
Table II lists all the events in the set Σ of the blockchain DES G (1), specifies whether each event is controllable (i.e., in Σ_c) or uncontrollable (i.e., in Σ_u), and indicates every constituent process the event is defined in. In what follows, these events and associated activities in a maintainer node are described, with reference to the node infrastructure depicted in Fig. 1 and their respective process models shown in Fig. 2. In the description, it is deemed as understood and so not explicitly stated that an event occurrence or execution is from a system state reached where the event is simultaneously defined at a state of every process model that it belongs to.

1) Event new_tx_rcd: Underlying, the incoming transaction queue continually stores incoming transaction messages (also concurrently received by the other maintainer nodes). This event is executed whenever a new transaction in the queue is fetched for validation.
2) Event tx_vdx_dn: Underlying, every transaction message fetched from the queue has been attached with a unique electronic signature. A single (-user) signature is the most basic, formed by hashing a transaction message and encrypting the hashed message using the single-user sender's private key at a sender node, where the transaction is initiated. This event is executed when the transaction validation is done (cryptographically), along with depositing the transaction into the self-block buffer if it is found to be valid, to gradually assemble the next self-validated block, and discarding the transaction otherwise. Transaction validation is done to check for transaction integrity, and this includes checking all the following:
   - That the transaction in the received message is original (i.e., not tampered with) and is indeed from the claimed sender. This entails a digital signature-based check[12], of which the most basic case is to verify a single-signature transaction, done by hashing the transaction

[12]Note that the Bitcoin application (of the founding blockchain) can support more complex transactions that require multiple signatures; the multi-signature-based check is formally specified in [22].

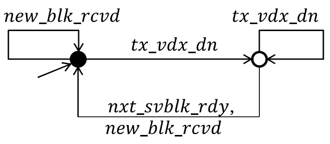


(a) TX_VALIDATION G_1.

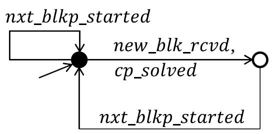


nxt_svblk_rdy, new_blk_rcvd
nxt_svblk_rdy, new_blk_rcvd
$nxt\_blkp\_started$
$vblk\_broadcasted$, $vblk\_chained$

(c) LEDGER_BLKUPDATE G_3.



(b) BLK_INPUT G_2.



(d) CONSENSUS_FIND G_4.

Fig. 2. Models of the blockchain operational processes.

TABLE II
BLOCKCHAIN EVENTS AND THE PROCESS MODELS (SEE FIG. 2) THEY ARE DEFINED IN.

Events - **c**: controllable; **u**: uncontrollable		TX_VALIDATION G_1	BLK_INPUT G_2	LEDGER_BLKUPDATE G_3	CONSENSUS_FIND G_4
new_tx_rcd	**c**	$\surd$			
tx_vdx_dn	**c**	$\surd$	$\surd$		
nxt_svblk_rdy	**u**		$\surd$	$\surd$	
new_blk_rcvd	**u**		$\surd$	$\surd$	$\surd$
$vblk\_broadcasted$	**c**			$\surd$	
$vblk\_chained$	**c**			$\surd$	
cp_solved	**c**				$\surd$
$nxt\_blkp\_started$	**c**			$\surd$	$\surd$

message received, decrypting the attached signature using the anonymous sender's public key, and comparing the hashed message with the decrypted signature for identity.

- That the transaction satisfies all the transaction validation rules defined, including rules for checking that the transaction is not the same as or not in conflict[13] with any transaction already recorded in currently the longest chain first formed in the ledger copy.

Depending on the application, this validation could also involve a request to an 'oracle', some supporting service that exists outside the blockchain - to verify application-dependent details associated with the transaction.

3) Event nxt_svblk_rdy: The self-block buffer is globally set to a block size limit (which is 1 MB for Bitcoin[14]), and a validated block is assembled once the validated transactions fill up the buffer and are sorted in serial order by their creation times. This event is executed when the self-block buffer, where the validated transactions are continually deposited, reaches some locally set level not exceeding the block size limit, and has the validated transactions sorted. The execution signals that the self-validated block assembly is ready for consideration as the next update for the local ledger copy and broadcast to the network.

4) Event new_blk_rcvd: Underlying, the received-block buffer continually stores each block received from the network if the block is found to be valid. The block is discarded otherwise. Block validation is done to check for block integrity, and this includes checking all the following:

- That the block is original (in content and sender) by a digital signature-based check.
- That the block's size does not exceed the limit set.
- That the block has the provided puzzle solution placed in its header verified to be correct for the consensus puzzle, set with a difficulty threshold given by that also stored in its header.
- That the timestamp of the block is valid.

Before executing the event, it is determined whether each new block in the received-block buffer is 'hash-chainable' to a block in the local ledger copy, i.e., whether a block in the ledger copy can be found that the received block can be hash-chained to. The block in the ledger copy is found for a received block provided the block ID, a hash code generated based on its block header, is equal to the previous block ID stored in the received block's header (making the block the received block's 'previous block' in the ledger copy once the received block is chained to it).

A block in the received-block buffer is said to be a

[13]In the case of Bitcoin, there is a 'no double spending' conflict-checking rule that stops the same Bitcoin from being spent more than once.

[14]Note that increasing the size limit of the transaction block has been a subject of much debate [23].

winning block of the current consensus race, if a chain in the current ledger copy remains or becomes solely the longest after it is appended with the block. Following, the event occurs if the received-block buffer contains a block that is ready for transfer to the local ledger copy, in that the buffer contains either a winning block of the current consensus race and a hash-chainable block, or a hash-chainable block while the self-block buffer has not accumulated enough validated transactions to create the next self-validated block.

5) Event $vblk\_broadcasted$: This event is executed when a self-assembled-and-validated block is broadcasted through some gossip protocol. Prior to broadcast, the following activities are completed in the following order:
   - Taking the end block of currently the longest chain first formed in the ledger copy (as it is the block to chain the assembled block to), the hash code of this end block's header is generated as the previous block ID, and placed in the block header along with the block timestamp and Merkle root.
   - The rest of the Merkle tree along with the block of transactions and auxiliary data (block size and number of transactions) are placed in the block body.
   - The block is chained to the local ledger copy.
   - The block is attached with a digital signature (generated herewith).

   Immediately upon the event execution, the self-block buffer is emptied (for next self-block processing).
6) Event $vblk\_chained$: This event is executed when a block stored in the received-block buffer that has been found to be hash-chainable is appended in the ledger copy. Its execution also implies that, along with discarding the appended block from the received-block buffer, if the appended block is a winning block of the current consensus race, the validated (user-initiated) transactions in the self-block buffer are all put back in order at the output end of the incoming message queue (for revalidation as needed in next self-block processing), and the self-block buffer is emptied.
7) Event cp_solved: This event occurs every time a consensus puzzle is solved, affirming a self-validated block as the next block to add to the ledger and to broadcast to the network next. The event occurrence also implies that the puzzle solution found as PoW for the self-validated block and the predetermined difficulty threshold $target(p)$ used are both placed in the block header.
8) Event $nxt\_blkp\_started$: This event is executed to signal the start of processing a new block (for ledger-update consideration as the next block to be placed) on top of currently the longest chain of blocks first formed in the ledger copy. If the chain extended by the last block addition is (solely) the longest, then the event execution also implies the confirmation (see Footnote 11), under ledger T-consistency, of the first of the last T consecutive blocks[15] in the longest chain formed. (For Bitcoin, $T = 6$.)

Note that, over a blockchain network, an honest maintainer broadcasts a validated block by executing its own event $vblk\_broadcasted$. However, a new block that it receives and validates, for transfer to its ledger copy when it asynchronously executes its own event new_blk_rcvd, is broadcasted by another maintainer that may be honest or dishonest.

C. The Specification Models

Over the local system DES G (1) in a network node, the collaborative ledger-update synchronization requirements that an honest maintainer needs to meet may now be formally specified. In essence, the overall specification prescribes temporally the orderly but competitive collaboration of the maintainer with other maintainers in the network, directing the local operations of when to append the block self-assembled and when to instead append the block received from another maintainer to its ledger copy, to keep it synchronized block by block with the ledger copies of all the other maintainers in the network. The competition for ledger block-update in every local block-processing cycle is between two events: cp_solved and new_blk_rcvd. This ledger-update synchronization specification is prescribed by a modular language $K = K_1 \cap K_2$. The (regular) constituent languages $K_1, K_2 \subseteq \Sigma^*$ are modeled, respectively, by trim automata R_1 and R_2 as shown in Fig. 3. Each of the specification automata is defined with the same event set Σ as the blockchain DES. Below, the specifications, K_1 and K_2, are informally described.

1) 'Take-My-Block' Specification, $K_1 = L_m(R_1)$: Modeled as shown in Fig. 3(a), K_1 requires that whenever the next self-assembled-and-validated block is ready, only then must a consensus puzzle be set and solved next, unless a new and validated block received (from some other maintainer in the network) is ready for transfer to the local ledger copy, in which case any initiated consensus puzzle solving is aborted. And if the puzzle set could be solved, then the self-assembled block will be appended to the local ledger copy and broadcasted (to all the other maintainers in the network) next, before the next block processing starts.

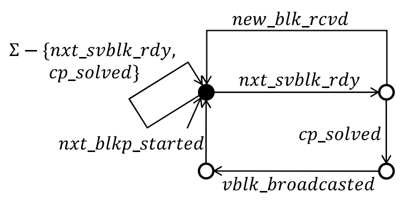
2) 'Take-Your-Block' Specification, $K_2 = L_m(R_2)$: Modeled as shown in Fig. 3(b), K_2 requires that whenever a new block is received (from some other maintainer in the network), validated, and ready for transfer, the local ledger copy must next be updated with this new validated block chained to it, before the next block processing starts.

Specifications K_1 and K_2 can also be thought of as jointly directing the intended flow of events in DES G (1) to streamline the underlying activities of the blockchain operations.

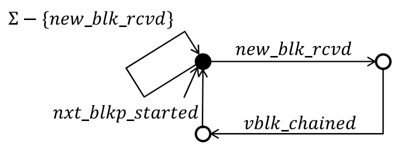
V. Supervisor Synthesis & Validation

Given DES G (1) and specification $K = K_1 \cap K_2$ (see Fig. 3), the control synthesis and validation of an honest

[15]Under honest power-majority, the PoW difficulty level can be calibrated to maintain ledger T-consistency at a feasible, relatively constant average rate of ledger block-update. At any juncture then when the last block addition extended a chain making it the longest chain (mandated as correct) to exist again in the local ledger copy, the longest chain's suffix, whose prefix's end block is where the latest forkings originated, is at most $(T-1)$ blocks long.



(a) Automaton R_1 for 'Take-My-Block' specification K_1.



(b) Automaton R_2 for 'Take-Your-Block' specification K_2.

Fig. 3. Specification models of blockchain update synchronization.

ledger-maintainer operational model is done using TCT [19]. In the discussions that follow, only the key TCT procedures are mentioned; auxiliary procedures used are omitted.

A. Model Synthesis

Using $Sync$, the blockchain DES G is created monolithically as a synchronous product of the constituent process models G_1, G_2, G_3, and G_4 (shown in Fig. 2). The computed monolithic G (not shown) has 24 states and 76 transitions, and is trim.

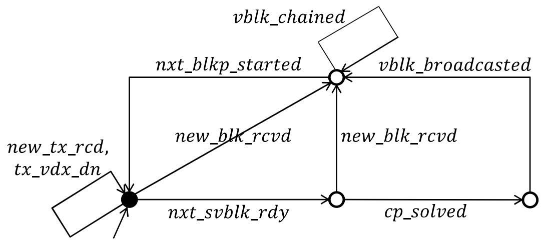


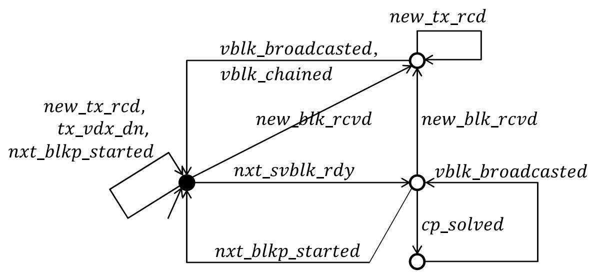
Fig. 4. State reduced model of monolithic supervisor S for specification $K = L_m(R_1) \cap L_m(R_2)$.

Expressed in conjunction with the constituent process models of the blockchain DES G (1), the full operational model obtained of an honest ledger-maintainer in monolithic-supervisor form is given by

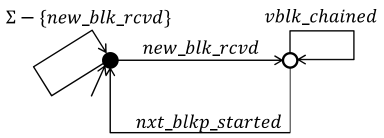
$$S \sqcap (G_1 \parallel G_2 \parallel G_3 \parallel G_4), \quad (2)$$

where the monolithic supervisor S therein is shown in Fig. 4 and can be obtained as follows: Using $Meet$, the Cartesian product R of R_1 and R_2 is computed, with no trimming required. The computed trim automaton $R = R_1 \sqcap R_2$ (not shown) is input as the model for $K = L_m(R_1) \cap L_m(R_2)$. Using $Supcon$ and $Supreduce$, the state reduced S, for which $S \sqcap G = Supcon(G, K)$, is obtained and found in fact to be state minimal.

Now, using $Supcon$ and $Supreduce$, each of state reduced S_i for $S_i \sqcap G = Supcon(G, K_i)$, $i = 1, 2$, can also be computed and both are found to be state minimal, as shown in Fig. 5. This is an interesting case demonstrating that, although $K_1 = L_m(R_1)$ and $K_2 = L_m(R_2)$ can be tested using $Condat$ to be controllable with respect to DES G, it turns out that $S_2 \sqcap G$ is equivalent to $R_2 \sqcap G$ (i.e., $S_2 \sqcap G = R_2 \sqcap G$) but $S_1 \sqcap G$ is *not* equivalent to $R_1 \sqcap G$ - the latter being counter-intuitive to a control nonspecialist. The fact is, for such equivalence to hold, the conditions are language controllability, which holds for K_1 and K_2, and a



(a) Supervisor S_1 for 'Take-My-Block' specification $K_1 = L_m(R_1)$.



(b) Supervisor S_2 for 'Take-Your-Block' specification $K_2 = L_m(R_2)$.

Fig. 5. State reduced models of constituent supervisors for $K = K_1 \cap K_2$, the specification of blockchain update synchronization.

positive $Nonconflict$ test outcome, which is so between R_2 and G, but not so between R_1 and G.

Using the $Nonconflict$ test on $S_1 \sqcap G$ and $S_2 \sqcap G$ - both trim because the respective $Supon(G, K_i)$, $i = 1, 2$, to which each reachable automaton is equivalent, is trim, $L_m(S_1 \sqcap G)$ and $L_m(S_2 \sqcap G)$ are proved to be nonconflicting. It follows by Theorem 1 that, equivalent to Control Model (2), an alternative model in modular-supervisor form is given by

$$(S_1 \sqcap S_2) \sqcap (G_1 \parallel G_2 \parallel G_3 \parallel G_4), \quad (3)$$

with S_1 and S_2 of Fig. 5 constituting the modular supervisor.

B. Model Validation

The foregoing control synthesis got the *design of the model right*, in that the marked language the ledger-maintainer model generates without blocking is guaranteed by construction using TCT to be within $K \cap L_m(G)$, where $K = L_m(R)$, with $R = R_1 \sqcap R_2$. What remains is to validate the design, i.e., to show that it is also *logically the right model*.

As textually described in [2], the blockchain's operational purpose is to have all maintainers on a DAO-T2Net doing ledger updating that is synchronizing to keep their ledger copies consistent. This is on the supposition that only honest ledger-maintainers consistently win the consensus race to have their newly assembled and validated blocks appended in the distributed global ledger. In determining if the model of an honest maintainer is logically valid for that purpose, under the stated supposition, the following questions are asked:

Q1) Can an honest maintainer properly assemble every block (of transactions) for ledger update?

Q2) Does an honest maintainer compete with other maintainers in the network in the way intended of the protocol, to win consensus for its block to be taken as the common block for the next ledger update?

Q3) Do the ledger-update actions of an honest maintainer follow the consensus?

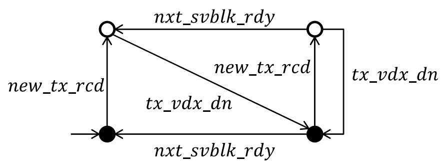


Fig. 6. BLK-PASM.

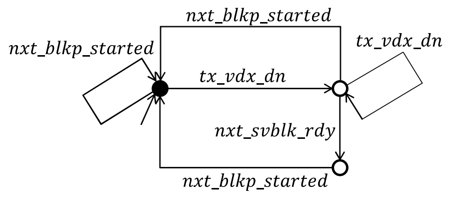


Fig. 7. NXT-BLKASM.

Q1 raises two behavioral aspects that an honest maintainer needs to abide by. These two aspects are formalized by automaton models BLK-PASM and NXT-BLKASM, as shown in Figs. 6 and 7, respectively. Abiding by BLK-PASM means that, from the initial state, tx_vdx_dn always occurs at least once before nxt_svblk_rdy can. This is necessary for at least one transaction from a network user to be stored in the self-block buffer, which holds the next self-validated block for ledger-update consideration when the block in the buffer is signaled as assembled and ready. Besides, the self-block buffer is never prevented from being filled with transactions, until a proper block assembly is signaled as ready in the buffer, with the process of fetching and validating a transaction (i.e., executing new_tx_rcd followed by tx_vdx_dn) permitted for as long as it is needed to assemble and get a block ready. Abiding by NXT-BLKASM means that, if an honest maintainer's block assembly is ready for update consideration (i.e., if nxt_svblk_rdy is executed), no further transaction will be validated and deposited in the self-block buffer (i.e., tx_vdx_dn is disabled), until after the next (new) block processing starts (i.e., until $nxt\_blkp\_started$ is executed). Abiding by these two models together means that every block can be properly assembled according to a specified block size, with no loss of transaction record due to overflow of transactions in the self-block buffer.

Q2 raises one behavioral aspect to abide by, which is formalized by automaton model CR4-MYBLK, as shown in Fig. 8. Abiding by CR4-MYBLK means that the competition to seek consensus is as intended: An honest maintainer can

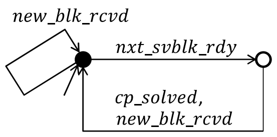


Fig. 8. CR4-MYBLK.

start solving a consensus puzzle (i.e., cp_solved is defined) only after its block is ready (i.e., only after nxt_svblk_rdy has occurred). By design, whoever in the network that solves a consensus puzzle first[16] wins the block-update consensus. Abiding by the model CR4-MYBLK, either the maintainer or some other from the network can win it, as signaled by cp_solved or new_blk_rcvd, respectively.

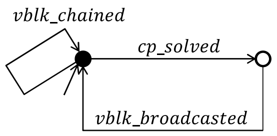


Fig. 9. WIA-MYBLK.

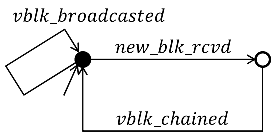


Fig. 10. LOA-YRBLK.

Q3 raises two behavioral aspects to abide by, which are formalized by automaton models WIA-MYBLK and LOA-YRBLK, as shown in Figs. 9 and 10, respectively. Abiding by WIA-MYBLK means that an honest maintainer will always update its own ledger copy next with its self-assembled block along with broadcasting the block to the network (i.e., $vblk\_broadcasted$ will be executed next), only if it has won the consensus (i.e., only if cp_solved is executed). Abiding by LOA-YRBLK means that the maintainer will always update its own ledger copy next with a block received from the network (i.e., $vblk\_chained$ will be executed next), only if it finds the received block valid and ready for transfer (i.e., only if new_blk_rcvd has occurred, indicating that some other maintainer in the network has won the consensus). Abiding by these two models together means that the ledger update by an honest maintainer follows the consensus reached each time.

The operational design space of the honest maintainer model is effectively modeled by automaton $GS = Supcon(G, K)$, where $K = L_m(R_1 \sqcap R_2)$. The computed GS (not shown) has 14 states and 22 transitions. It then follows that answering 'Yes' to each of Qs. 1 to 3 regarding the honest maintainer model is about formally showing that every aspect raised by

[16]It is appropriate at this juncture to clarify that *being first in solving a consensus puzzle* is a local, not global notion. A maintainer is deemed to have solved a consensus puzzle first if it does so before it receives and validates a new block from the network for transfer to its ledger copy. It is possible, though not intended, that two or more maintainers solve their puzzle at about the same time.

the question is correctly matched by a projection of the trim automaton GS, using $Project$, that retains the design space containing only the subset of events relevant to the aspect. The affirmative answers are stated in Theorems 2 to 4.

Theorem 2: Q1 - Yes, i.e., an honest maintainer can properly assemble every block for ledger update.

Proof: $Project(GS, H_1)$ = BLK-PASM (in Fig. 6), where $H_1 = \{new\_tx\_rcd, tx\_vdx\_dn, nxt\_svblk\_rdy\}$. $Project(GS, H_2)$ = NXT-BLKASM (in Fig. 7), where $H_2 = \{tx\_vdx\_dn, nxt\_svblk\_rdy, nxt\_blkp\_started\}$. ■

Theorem 3: Q2 - Yes, i.e., an honest maintainer competes with other maintainers in the network to win consensus in the way intended.

Proof: $Project(GS, H_3)$ = CR4-MYBLK (in Fig. 8), where $H_3 = \{nxt\_svblk\_rdy, new\_blk\_rcvd, cp\_solved\}$. ■

Theorem 4: Q3 - Yes, i.e., the ledger-update actions of an honest maintainer do follow the consensus.

Proof: $Project(GS, H_4)$ = WIA-MYBLK (in Fig. 9), where $H_4 = \{cp\_solved, vblk\_broadcasted, vblk\_chained\}$. $Project(GS, H_5)$ = LOA-YRBLK (in Fig. 10), where $H_5 = \{new\_blk\_rcvd, vblk\_chained, vblk\_broadcasted\}$. ■

By Theorems 2 to 4, the validity of the honest maintainer model designed is determined. This means that it is the right logical model - one that fulfills the working order intended of an arbitrary honest ledger-maintainer on a DAO-T2Net.

With hindsight, one might find the validation results to be obvious and expected. However, as with any formal approach, it is essential to formally confirm these, to ensure that no unintended constraint imposition or relaxation is inadvertently introduced in the system and specification modeling stages.

VI. Discussion with Related Work

A Universal Composition (UC) model [1], [24], a Script-abstracted transactions model [22], and a Markov decision process (MDP) model [4] are among the analysis models that have been developed for studying various performance aspects of the Satoshi Nakamoto blockchain with regard to its desired properties and security. Several security risks (against attacks) have also been identified, assessed, and mitigated or rationalized [25], [26], [27], [28]. In contrast, the contribution of this paper is a new logical, operational control model of an honest ledger-maintainer resident in a distributed node of the Satoshi Nakamoto blockchain network. This model is provably assured of correct blockchain operations against qualitative ledger-update synchronization specifications, and is validated. Existing performance analysis models attempt to, in one way or another, represent and evaluate the overall runtime operational behavior of the blockchain in a possibly adversarial network environment. This runtime behavior, in turn, is the collective outcome of every honest ledger-maintainer working continually alongside transaction network users and adversaries (i.e., dishonest ledger-maintainers). The local working of an honest maintainer is, hitherto, not formally modeled with explicit structural information that can be readily captured by the system concept - that of events in an event-based model. In the opinion of this paper, the formally derived operational control model from the proposed discrete-event control-theoretic formulation fills this gap, providing an unprecedented logical systematization of operational knowledge of the Satoshi Nakamoto blockchain. In essence, the nonblocking subspace defined by the logical model is operationally invariant and holistic. Decidedly, the role that the event-based structure of this model plays is in blockchain system engineering and research, where simplicity and clarity dominate over accuracy and detail, and are facilitated by model determinism, in the sense that from every model state, a transition by the same event always leads to the same state.

Importantly, the potential impact of the logical model lies in its being able to unify the foundational aspects of the Satoshi Nakamoto blockchain, exhibited by Figs. 6 through 10, into an implementation-independent reference that is understandable and explainable for system engineering and research of blockchains. For systems and control specialists, the common intellectual understanding it could foster should direct a more unified research of security and performance issues underlying the logical blockchain operations. This should bring about high confidence development of and more consistent security risk identification, assessment, and mitigation across the many emerging real-world decentralized applications of blockchain technology, i.e., the application DAO-T2Nets. These application DAO-T2Nets, for decentralized business and service management, are either new or could displace or transform their legacy counterparts that require middleman involvement. Besides finance, credentials, and supply chain logistics, the applications could include personalized healthcare [29], intelligent robots and drones as Internet-of-Things [10] to provide autonomous transport services such as package delivery [30], and a lot more.

VII. Conclusion

This paper has systematically developed and validated an operational discrete-event control model, in either monolithic form (2) or modular form (3), that is generic of every honest ledger-maintainer whose role is central in blockchain technology based on the PoW-based consensus [2]. In the process, it has introduced and demonstrated an effective use of formal methods from supervisory control theory [12] in guiding model design synthesis and validation, geared towards a control science of blockchains. In 'extracting' logical simplicity, of blockchain operations as local control of a system resident in a network node, in turn, it is hoped that the formal model would be widely adopted as a system engineering and research reference by the blockchain community, to help bring about the secure and consistent development of many application DAO-T2Nets. For supervisory control theorists, it is hoped that this novel application of the theory would inspire new decentralized control ideas for discrete-event control synthesis of asynchronous Internet systems in general.

This paper has developed models rendering blockchain networks amenable to *discrete-event systems and control* thinking and methods. Along this fresh direction, one future work of theoretical interest is evolving and augmenting the maintainer control model developed in this paper with critical timing features using a real-time version of supervisory control theory

[31]. Another is discrete-event control modeling of blockchain ledger-maintainers on markedly different consensus protocols [8] for building DAO-T2Nets, such as those based on proof of stake [32], [33].

References

[1] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Lecture Notes in Computer Science: Advances in Cryptology - EUROCRYPT 2017, Vol. 10211*, J.-S. Coron and J. B. Nielsen, Eds. Springer, Cham, 2017, pp. 643–673.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[3] IEEE Spectrum, "Blockchain world," October 2017. [Online]. Available: https://spectrum.ieee.org/static/special-report-blockchain-world

[4] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Čapkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)*, Vienna, Austria, October 2016, pp. 3–16.

[5] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Lecture Notes in Computer Science: Advances in Cryptology CRYPTO 1992, Vol. 740*, E. F. Brickell, Ed. Springer, Berlin, Heidelberg, 1993, pp. 139–147.

[6] B. Warburg, "How the blockchain will radically transform the economy," June 2016. [Online TED Lecture]. Available: https://www.ted.com/talks

[7] M. Conti, S. Kumar E, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," 2017. [Online]. Available: https://arxiv.org/abs/1706.00916

[8] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, March 2016.

[9] Nomura Research Institute, "Survey on blockchain technologies and related services," Japan's Ministry of Economy, Trade and Industry (METI), FY2015 Report of Contract Survey, March 2016. [Online]. Available: http://www.meti.go.jp/english/press/2016/pdf/0531_01f.pdf

[10] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, techniques, and applications," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, vol. 48, no. 9, pp. 1421–1428, September 2018.

[11] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206–230, January 1987.

[12] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, A. Isidori, J. H. van Schuppen, E. D. Sontag, and M. Krstic, Eds. Springer International Publishing, 2019.

[13] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, MA : Addison-Wesley, 1979.

[15] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal of Control and Optimization*, vol. 25, no. 3, pp. 637–659, May 1987.

[16] R. Su and W. M. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Systems : Theory and Applications*, vol. 14, no. 1, pp. 31–53, 2004.

[17] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of Control, Signals and Systems*, vol. 1, no. 1, pp. 13–30, January 1988.

[18] C. G. Cassandras and S. Lafortune, "Ch 2 : Languages and Automata," in *Introduction to Discrete Event Systems*, 2nd ed. Springer-Verlag, New York, 2008, pp. 53–132.

[19] W. M. Wonham, *Control Design Software Tool: TCT*. Developed by Systems Control Group, University of Toronto, Toronto, ON, Canada, May 2017. [Online]. Available: http://www.control.utoronto.ca/DES/Research.html

[20] Technical Committee on Discrete Event Systems (DESTC), "List of software tools for discrete-event control design." [Online]. Available: http://discrete-event-systems.ieeecss.org/tc-discrete/resources (As of November 2017).

[21] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Lecture Notes in Computer Science: Advances in Cryptology CRYPTO 1987, Vol. 293*, C. Pomerance, Ed. Springer, Berlin, Heidelberg, 1988, pp. 369–378.

[22] N. Atzei, M. Bartoletti, S. Lande, and R. Zunino, "A formal model of bitcoin transactions," Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC 2018), Santa Barbara Beach Resort, Curaçao, 2018. [Online]. Available: https://fc18.ifca.ai/preproceedings/92.pdf

[23] G. Andresen, "Block size limit controversy." [Online]. Available: https://en.bitcoin.it/wiki/Block_size_limit_controversy (Accessed November 2017).

[24] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *Proceedings of the 31st International Symposium on Distributed Computing (DISC 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), A. W. Richa, Ed., vol. 91. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 39:1–39:16.

[25] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Proceedings of the 18th International Conference on Financial Cryptography and Data Security (FC 2014)*, ser. Lecture Notes in Computer Science, N. Christin and R. Safavi-Naini, Eds., vol. 8437. Christ Church, Barbados: Springer, Berlin, Heidelberg, March 2014, pp. 436–454.

[26] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C., USA: USENIX Association, August 2015, pp. 129–144.

[27] I. Eyal, "The miner's dilemma," in *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P 2015)*, San Jose, CA, USA, May 2015, pp. 89–103.

[28] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *Proceedings of the 1st IEEE European Symposium on Security and Privacy (Euro S&P 2016)*, Saarbrücken, Germany, March 2016, pp. 305–320.

[29] P. Yang, D. Stankevicius, V. Marozas, Z. Deng, E. Liu, A. Lukosevicius, F. Dong, L. Xu, and G. Min, "Lifelogging data validation model for Internet of Things enabled personalized healthcare," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, vol. 48, no. 1, pp. 50–64, January 2018.

[30] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, January 2017.

[31] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–341, February 1994.

[32] I. Bentov, A. Gabizon, and A. Mizrah, "Cryptocurrencies without proof of work," in *Lecture Notes in Computer Science: Financial Cryptography and Data Security FC 2016, Vol. 9604*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Springer, Berlin, Heidelberg, 2016, pp. 142–157.

[33] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Lecture Notes in Computer Science: Advances in Cryptology CRYPTO 2017, Vol. 10401*, J. Katz and H. Shacham, Eds. Springer, Cham, 2017, pp. 357–388.