The consensus machine: formalising consensus in the presence of malign agents

A. W. Roscoe[1,2,3], Pedro Antonino[1], and Jonathan Lawrence[1]

[1] The Blockhouse Technology Ltd., Oxford, UK
[2] Department of Computer Science, University of Oxford, Oxford, UK
[3] University College Oxford Blockchain Research Centre, Oxford, UK
awroscoe@gmail.com, pedro@tbtl.com, jonathan@tbtl.com

Abstract. We show how a group of independent agents with carefully specified notions of progress and synchronisation can proceed in a model where consensus does not require unanimity and represents an emergent group behaviour. These agents will typically cooperate to implement a decentralised system such as a blockchain. Thus a model state machine can be compiled into replicated asynchronous agents that implement it in such a way as to overcome Byzantine faults up to a certain assumed level.
Our abstraction achieves precise reliability and allows us to create schemes in which multiple consensus machines interact that can be modelled in process algebras like CSP and verified by model checking. This paper concentrates on the CSP analysis of a protocol showing how one distributed consensus machine can take over control from another without creating ambiguity over the outcome.

Keywords: Consensus · State machine · Byzantine · Process algebra · Formal methods.

1 Introduction

In this and related papers we show how to navigate the giant conceptual leap from a simple trusted state machine acting dependably on the collective data of a distributed system, which we term the *ideal*, to an implementation of the same functionality on a collection of decentralised agents that are neither universally trustworthy nor universally trusting. Instead we create a model of this trust and dependability which is frequently stochastic. We show how to create rules which, for suitable models, deliver a decentralised implementation of the desired behaviour and allow the creation of new and efficient consensus protocols.

The core example of this is a public blockchain. In this we understand what high level procedure is meant to be followed: the blockchain selects successive agents to create blocks; it verifies the ones put forward and either accepts or rejects them. The rules for these procedures are laid down and universally recognised. It is quite normal to explain such actions by the chain as being the work of a single god-like agent, and frankly such understanding provides the best high

level and relatively elementary explanation of what is meant to be going on. However the behaviour has to emerge from the actions of many agents purporting to act together, when there may be some among them who deliberately seek to undermine this process.

As well as creating a theory of consensus, we use it to understand and justify a new model of block approval/verification for blockchains. We term this the *picketing model*, which we summarise informally here:

1. We note that it is unfortunate if a consensus model builds in the assumption that it has to counteract a large proportion of active attackers all the time, when mostly the bad agents — the corruptible ones — can be expected to follow the rules because of incentive structures. This means that a lot of machinery needs to be present to counteract attacks that are usually not there.
2. Our proposal to get around this is to use a primary decision mechanism that will demonstrably come to the right decisions if most of the bad agents make the same choices as the good ones, but which may fail to come to any decision if they do not. We ensure that bad agents can never force a bad decision.
3. This is backed up by a secondary decision-making mechanism that is less efficient but cannot be blocked by the bad agents under standard assumptions. This takes over if the primary mechanism apparently fails to make a decision.

Conceptually this seems relatively clear. The problems come from getting it to work securely in the decentralised world of agents, some of whom are malign. We identify the following issues:

A. Creating a safe but not necessarily complete model of consensus. This means identifying a set of *pickets* (i.e. block-producing agents) upon whom all agree. Moreover, it means coming up with a model of when there is sufficient consensus among these so that all will agree when this is demonstrated, and similarly agree that no commitment exists without this.
B. Here, by safe we mean that there cannot be consensus without this being agreed by sufficient agents — generally good agents. We need a universally agreed definition of this so consensus is equally recognised by all. Similarly it is impossible for two conflicting consensuses to arise about the same question.
C. Ensuring that enough good agents exist for progress to be near-certain unless bad ones misbehave in an obvious way.
D. Understanding how bad agents can prevent progress by preventing consensus. This can arise from them certifying wrong answers or failing to give any answer at all.
E. How to build a safe and complete back-up mechanism. We would expect the completeness to come from involving many more agents in the process so that it is effectively impossible for there to be enough bad ones to block progress. This might well just be Byzantine agreement over the complete mining population. We will find that the combinatorics of building a safe

and complete system are a natural — and more demanding — extension of building a safe one.

F. As soon as there are more than one decision making mechanism in a decentralised and asynchronous environment, we need to prove that they cannot make inconsistent decisions. For example, if they are given the task of deciding on the next block in combination and expected to come up with a decision, we need to show that at least one of them does and if they both do they agree. For otherwise this might lead to a fork.

2 Background

2.1 Blockchains

Blockchains were initially proposed as a decentralised way to implement digital currencies and prevent double spending, i.e. the possibility that the owner of a digital currency could spend it more than once [20]. However, they have evolved into generic decentralised auditing systems that do much more than just prevent double spending. For instance, with the advent of smart contracts — programs that are executed in the context of a blockchain — a developer can define by means of a program how transactions addressed to that smart contract are to be processed [5].

A blockchain is a *decentralised stateful transaction processing system*, sometimes referred to also as a *distributed ledger*. It receives transactions from its stakeholders, decides on which of those are valid, and performs alterations to its state that record the effects of these transactions. As a decentralised system, multiple agents collaborate to implement this behaviour. In this context, the term blockchain refers not just to the state comprising the transactions and blocks, but includes the entire system including the agents operating on the state.

A blockchain orders and stores valid transactions into *blocks* which are themselves ordered, giving rise, ultimately, to a *chain of blocks* representing the history of the blockchain. In practice, however, during its operation, a blockchain — or rather its agents — manipulate a *block tree*.

A block tree is a directed, finite and acyclic rooted tree defined by a pair (V, E) where V is a set of blocks and E is a set of backward links — the root (genesis block) is the only block without an outgoing link. The backward links are implemented by embedding the cryptographic hash of its predecessor block in the header of each block. A backward link exists from block B_2 to block B_1 *iff* $hash\_pointer(B_2) = hash(B_1)$, where $hash\_pointer()$ extracts the embedded backward link from a block, and $hash()$ is the cryptographic hash function used by the blockchain. This results in a unique path from every block back to the root (B_0), because (by the properties of the hash function) it is infeasible to construct a false predecessor block with the same hash.

However, because it is possible to construct many different valid successor blocks to any existing block, it is necessary for the blockchain's agents to have a mechanism to determine unambiguously which is the "true" successor to any

given block. It is an aspect of the design of a reliable consensus mechanism to achieve this that is the motivation for this paper.

2.2 CSP and its semantics

Note: In this paper we use the machine-readable ascii version of CSP syntax (CSP_M) throughout, as opposed to the typeset blackboard syntax and symbols.

CSP is based on instantaneous actions handshaken between a process and its environment, whether that environment consists of processes it is interacting with or some notional external observer. It enables the modelling and analysis of patterns of interaction. The books [12, 24, 27] all provide thorough introductions to CSP. The main constructs that we will be using in this paper are set out below.

- The processes `STOP`, `SKIP` and `DIV` respectively do nothing, terminate immediately with the signal ✓ and diverge by repeating the internal action τ. `RUN(A)` and `CHAOS(A)` can each perform any sequence of events from A, but while `RUN(A)` always offers the environment every member of `A`, `CHAOS(A)` can nondeterministically choose to offer just those members of `A` it selects, including none at all.
- `a -> P` *prefixes* `P` with the single communication `a` which belongs to the set Σ of normal visible communications. Similarly `[] x : A @ x -> P(x)` (replicated external choice) offers a choice over `A` and then behaves accordingly.
- CSP has several *choice* operators. `P [] Q` and `P |~| Q` respectively offer the environment the first visible events of `P` and `Q`, and make an internal decision via τ actions whether to behave like `P` or `Q`.
  The asymmetric choice operator `P [> Q` offers the initial visible choices of `P` until it performs a τ action and opts to behave like `Q`. In the cases of `P [] Q` and `P [> Q`, the subsequent behaviour depends on what initial action occurs.
- `P \ X` (hiding) behaves like `P` except that all actions in `X` become (internal and invisible) τs.
- `P [[ R ]]` (renaming) behaves like `P` except that whenever `P` performs an action `a`, the *renamed* process must perform some `b` that is related to `a` under the relation `R`. `R` is specified using the CSP_M mapping syntax.
- `P [| A |] Q` is a *parallel* operator under which `P` and `Q` act independently except that they have to agree (i.e. synchronise or handshake) on all communications in `A`. A number of other parallel operators can be defined in terms of this, including `P ||| Q = P [||] Q` in which no synchronisation happens at all.

There are also other operators such as `P ; Q` (sequential composition), `P /\ Q` (interrupt) and `P [| A |> Q` (throwing an exception) for passing control from one process `P` to a second one. `P /\ Q` hands over control when `Q` performs a visible action, so that the handover if instigated by Q. In `P [| A |> Q` it is instigated by `P` performing an exception event `a` from the set `A`.

It is always asserted that the meaning, or semantics, of a CSP process is the pattern of externally visible communication it exhibits. As shown in [24, 27], CSP has several styles of semantics, that can be shown to be appropriately consistent with one another. In this paper, we are concerned with *behavioural* semantics: CSP processes are identified with sets of observations that might be made from the outside. The best known behavioural models of CSP are based on the following types of observation: *Traces* are sequences of visible communications a process can perform. *Failures* are combinations (s, X) of a finite trace s and a set of actions that the process can refuse in a *stable* state reachable on s. A state is stable if it cannot perform τ. *Divergences* are traces after which the process can perform an infinite uninterrupted sequence of τ actions, in other words diverge. The models are then:

- `T` in which a process is identified with its set of finite traces;
- `F` in which it is modelled by its (stable) failures and finite traces;
- `FD` in which it is modelled by its sets of failures and divergences, both extended by all extensions of divergences: it is *divergence strict*.

2.3 FDR

FDR[1, 24, 27, 8] is a refinement checker between finite-state processes defined in CSP. First created in the early 1990's it has been regularly updated since. The latest version is FDR4.[4]

It uses CSP_M, the machine-readable version of CSP, which has been extended with a functional programming language related to Haskell. This enables the user to define complex networks and data operations succinctly, and to create functions that, given abstract representations of structures or systems, can automatically generate CSP networks to implement and check them. Perhaps the best-known example of this is the Security Protocol checker Casper [19] which, given an abstract representation of a cryptographic protocol and some security objectives for it, generates a CSP script which checks to see if the objectives are true. In a similar vein compilers have been written from other notations to CSP such as Statecharts [11] and shared-variable programs (see Chapters 18 and 19 of [27]). A survey of the most important practical applications of FDR can be found in [3].

FDR is most often used to check refinements of the form `Spec [X= Impl`, where `Spec` is a process representing a specification in one of the standard CSP models `X`, usually traces, stable failures or failures-divergences. `Impl` is a CSP representation of the system being checked. To check whether a process `Impl` satisfies a particular property, `Spec` is constructed to represent the most general process (in the relevant model) exhibiting the required property.

FDR supports a number of techniques for attacking the state explosion problem, including hierarchical compression and symmetry reduction [9]. The algorithms underpinning FDR are set out in [24, 27, 8, 25].

[4] Available at https://cocotec.io/fdr/.

3 Living in a decentralised world

As a blockchain develops, we assume that at any time the set of legitimate members is unambiguous — in developing block $n+1$ as a successor to the known block n, the set of members and their rights and signature method are agreed by all as a function of the chain to that point. These are the blockchain *agents*. Similarly the rules on how to go about picking the next block are laid down as is the target schedule for producing it. All agents have reasonably accurate clocks. To be accepted, each message between the agents needs to be signed and have a timestamp.

There is no synchronisation in such systems and yet we need the decentralised system implementing a blockchain to step forward in steps that are recognisable. This needs to be decisive and unambiguous despite the presence of malign agents.

We need to distinguish what we, as a privileged and god-like external observer see by examining the states of agents and what is communicated from the outside, and what is visible to agents operating in the collective system. Any agent will in some sense know whether it is good or bad, and good agents will not know which others are unless they have done something that marks them out as bad. We assume that the bad agents are in league with one other.

We will create a model in which the state of the system is derived from assertions that have been made by a designated collection of agents, namely the *pickets*, and which of these the agents have seen. The state will be designated by the external observer and the system will pass through a sequence tr of these.

Inevitably and unavoidably, agents themselves do not know for certain what state the system is in because it could have advanced beyond the most recent state for which it has evidence. Rather, we hope that each good agent has evidence of some member of tr and is thus either up-to-date or is lagging behind. In other words, it should be an invariant that each good agent — whether a picket or not — has its current state set to a member of tr.

We will describe stronger invariants for pickets in the next section.

The main contribution of this series of papers is an abstraction, the consensus machine, which allows groups of undependable and untrustworthy agents to implement such a high-level ideal behaviour. This takes the established concept of consensus by Byzantine agreement and generalises it, formalising it in a model derived from process algebra.

The essence of this is an exact formulation of consensus in a group of agents, each supposedly replicating what the ideal is meant to do. This is done in such a way that, despite whatever level of malign behaviour is assumed to be possible and the design allows for, it is the ideal behaviour that emerges.

In the world of blockchains this allows us to develop varying consensus algorithms, including ones that erode the distinction between public and coalition chains. By building in exception handling mechanisms we are able to build blockchains that can benefit from incentivising good behaviour from bad agents without relying on it. This paper concentrates on the proof of a protocol that ensures that this exception handling does not create any inaccuracy or ambiguity in decision making.

To summarise: no one other than those with the god-like ability to see into the states of all agents, and see their communications, can see the latest state of a decentralised system implemented by a set of pickets. However, our aim is to ensure that the state seen by the any component or external observer is always, if not the global state, is one of its predecessors.

4 Keeping your pickets in a row

In this and the next section we show how to achieve this goal when a single set of pickets is given responsibility. The minimal requirement of a set of pickets is that if they all agree on a decision then everyone good, and all relevant regulators, are satisfied that it is *the* decision, or at least a correct one that it is reasonable for all to agree.

In general it may be unreasonable to force all to agree, and we expect that in many cases sets of pickets will be selected where smaller agreeing groups are sufficient.

For any set P of pickets, and each type of decision D, there will be a decision set M of subsets of P such that agreement by any of these sets, or a superset, commits the system to the agreed decision. We can therefore either assume that M is a nonempty antichain of subsets of P or is superset closed and contains P. The determination of what represents a member of M will be built into the rules of the blockchain so there is no ambiguity about it: if a collection of pickets that agree to a decision is presented, good agents will never disagree about whether this commits the chain or not. In the following we use the superset closed version.

The criteria for what represents a valid M should depend on mathematical analysis in at least one risk model, and may well depend on the requirements of regulators in one or more jurisdiction. They might well depend on the nature of the decision D and the jurisdictions involved.

It may also depend on whether it is conceivable that two different good nodes might give different answers when asked for a decision on a particular question. This may well be the difference between a clear cut decision about whether something is valid or not, and a selection. No valid M can have two distinct members that might select two different decisions in the same state.

In a world without jurisdictions and just good and bad agents, then unless a secure communications framework prevents a bad node from communicating different decisions to different parties, an unambiguous decision will require more than half the good agents to agree with it. We will revisit this in the section on stochastic inference.

In what follows we will assume that the analysis behind each M is accurate so that ambiguity can never arise.

It is crucial to realise that we anticipate that reaching the hurdle of a member of M will often require bad agents to agree: it may not be possible to reach this hurdle without them. Understanding this is the key to this paper. Decisions are safe because ones that are made cannot lead to bad conclusions. They may be

incomplete because if the bad agents do not co-operate, no decision may be reached at all.

5 The unitary consensus machine

By this we mean a structure for computing on one set of pickets. The set of pickets P has been fixed, and we have an agreed correct mechanism for computing the decision sets M. The purpose of the consensus machine is to reliably compute the results of a high level state machine running against the background of our assumption of malign agents being present.

The high level state machine HS is described in the following language:

a. A state consists of a control state label plus an accumulation of data, plus output symbols. It also has an accurate clock
b. A state initially collects data. Once this is complete, potentially indicated by its clock it moves to its compute phase.
c. It computes from its state and this data both an output and a new state.

This state machine represents the intended emergent behaviour of the consensus machine.

This is compiled into a state machine that runs on each picket. This runs the same state machine as HS except for the following additions:

1. Each state is labelled with the trace of actions that brought it there. Thus, if the original machine can revisit a state, these visits are now treated as different states.
2. It expects its inputs to be signed and checks these.
3. It signs its own decision: the outputs and proposed next state, possibly coupled with a hash of the state from which these were computed.
4. There is a mechanism for aligning these input states: this might be agreement before the outputs are computed or a reaction to disagreement apparent when comparing outputs.
5. The picket reads the outputs coming from other pickets and publishes a certificate when sufficient agree that the set is in the M for its current state. This certificate contains the signed evidence of agreement and identifies the state transition it applies to. It should contain information that allows a picket seeing it to reconstruct its input state — e.g. the sources of that state information.
6. The component machine changes states when it either creates or sees a valid certificate for the state change, when this takes it to a later state than the one it is now in. It ensures that its data is that of the new state.

In other words it is the demonstration of a valid certificate that represents the state change in the decentralised machine. This could be anywhere including in a bad agent. Just because the certificate exists does not mean that everyone knows it. It follows that the states of the individual pickets are either the same as the whole machine or a predecessor of that.

The state of the unitary consensus machine (UCM) can be characterised by the sequence of certificates that has arisen and the most recent certificates seen by each good agent, together with all signed statements about the next action. It would be possible to create a full operational semantics in which each good agent runs through the component state machine and has its clock, while each bad agent's state and capabilities are essentially the same as the classic cryptoprotocol intruder.

For this to work we require that there can never be a situation where a given state can give rise to two inconsistent certificates: ones for different successor labels or different outputs. This comes down to the accuracy of the models supporting the construction of M. We also require that every state and output is consistent with HS, the high-level target. This depends on the accuracy of M, in particular on the presence of a good agent amongst those agreeing on each transition and the consistency of the data collection and the data state between the two.

Though not necessary for safety, the determinism of the calculations underlying HS is also important, because if all components of UCM are behaving properly we would expect that it will reach a valid conclusion. If there are more than one possible result thanks to nondeterminism then they might be split meaning that the threshold implied by M is not reached.

It is quite likely that multiple certificates arise for the same high level state change, simply because different collections of pickets satisfying M are seen to agree on the same thing.

Blockchains are frequently set up with incentive and penalty structures that are designed to persuade the bad to follow the rules. We categorise bad behaviour as follows:

1. Overt bad behaviour. Making contributions to the central discussions and protocols of a chain or other decentralised system that will be seen and recognised as bad. Unless this wins votes or similar, it will quickly recognised and perpetrator punished.
2. Covert bad behaviour. Producing non-compliant structures that are kept hidden and only perhaps revealed later. For example developing a fork alongside the true chain.
3. Non-participation. Failing to make contributions that are expected of a good agent and thereby denying some correct action the majority it needs. The main issues with this is that it is harder to penalise because a good agent may encounter communication failures, a phenomenon that can also mean confusion about how an apparently non-participating agent should be interpreted. It is fairly standard to make gossiping assumptions about communications in blockchains to resolve such confusion.

We will later show how to build hierarchical consensus machines, where control of decision making passes from one group of pickets to a back up mechanism.

6 Stochastic inference

It is natural to use probability to assemble sets of pickets and produce decision thresholds M. In this section we discuss a central case of how this can support the picketing model. We will assume the blockchain can select pickets independently and randomly from a given population, so that the number of good and bad ones that make any decision is governed by a binomial distribution governed by the number of pickets and the proportion of good agents in the population, weighted the same way as the random choice.

It is wise to chose a mechanism for this that guards against sharding: bad agents creating clones of themselves in an attempt to get chosen more often. One such class of mechanisms is proof of stake where agents are randomly selected with repetition: in choosing multiple agents, it is possible for the same one to be chosen more than once; the $k+1$th agent is chosen independently of who was chosen in the first k.

If there are any bad agents around, it is possible that all the pickets chosen are bad, just as throwing a 6-sided dice 100 times might give a sequence of 100 sixes. Rather it is almost certain that the distribution of a 100 picket choices or the numbers thrown will be a lot closer to the expected values: $p \cdot 100$ or one-sixth each, where p is the proportion of the population who are bad. It is relatively easy, when N pickets are chosen, to compute how likely it is that more than r are bad.

To exploit this we introduce the idea of *stochastic impossibility*: an event so unlikely that in the whole history of a system it is very unlikely ever to happen to the extent that it can be disregarded. One might regard a one-in-a-million chance as small enough, but if many (say a million) choices are going to be made a year (approximately one every 30 seconds) it is clearly is not enough if a single one can corrupt a system.

We believe that $\epsilon = 10^{-18}$ or, in terms of the normal distribution, 9σ (close to 10^{-19} as a one-sided extreme: we are only interested in unusually many bad agents, not good or bad), where σ is the standard deviation, is a good starting point for stochastic certainty: the chance of a single choice going wrong being this small. Someone might argue that it is not small enough because, for example, the puzzles in bitcoin proof-of-work protocol (PoW) have a probability of about 10^{-24} and they are solved every 10 minutes on average. But in that world there is no limit on the number of attempts to solve the puzzle and so one would need a much larger number for effective impossibility: the combinatorics would probably be the same as second preimage analysis in hash functions. In the world we are thinking of, a system places a bound (e.g. a million per year) on the number of trials. What this emphasises is that in assessing the requirements on the unlikelihood of a single trial, the context in which it arises is really important.

In a strong sense the suggestion of 10^{-18} is about bounding the chance of anything going wrong in a series of stochastic experiments conducted over time by a well behaved party who conducts the experiments at a few times the rate at which the data structure (e.g. blockchain) is incremented. It is only a suggestion.

In general terms the criteria by which pickets are chosen and agents make decisions must be agreed and public.

7 Stochastic decisions

From now on we concentrate on decisions based on assumptions about the preponderance of reliable good behaviour amongst the participants who are entitled to participate in various actions. In particular we examine the case where all of a population from which a group of pickets — delegated decision makers — are picked are deemed equally likely to behave well. Specifically we assume that one chosen at random has a chance bounded above by p of being bad or corruptible. Little of what follows would change without the constant p assumption except that the calculations would be more complex and potentially depend on exactly who was chosen.

Our assumption means we can calculate on familiar territory.

Recall that Byzantine agreement only works when less than $1/3$ of the voting population is bad. Several of the options below require similar. We will therefore normally assume $p < 1/3$, but will remark when this is not necessary.

We can now understand how to create the decision thresholds M described earlier. In a group of pickets P, a subset G represents a safe (or dominated) agreement if it is stochastically impossible that they are all bad. Thus if all members of G agree on a decision, that decision must be a reasonable one. It represents a safe (or dominated) strong agreement if it is guaranteed to contain at least half the good members of P. Depending on the circumstances of the decision, other thresholds may be appropriate.

Some sets of pickets will be too small, and even unanimity will not give a safe decision. Otherwise the above allows us to calculate the agreements that create valid certificates as described earlier. The appendix to this paper contains some examples of these calculations. There is further illustration of this style of stochastic reasoning in [28].

7.1 Who should we rely on?

Usually our systems do rely on decisions being made, and usually the systems are more efficient if they can persuade bad participants to contribute mostly as though they were good. Indeed for a consensus machine with a smaller set of pickets to deliver results, this is necessary. To achieve this they need three things: firstly strong incentives on agents not to misbehave and to participate constructively, secondly a decision making mechanism that prevents the bad from inducing a bad decision, and thirdly a fallback mechanism that can force correct decisions when needed, all be it at the cost of lower efficiency.

The last of these should convince opponents that they will not be able to permanently disrupt the system, for example by getting a blockchain to fork. The worst they can achieve is complication and delay. One cannot reasonably

prevent the bad from covert mischief, but overtly (in the sense that their non-standard action or inaction is identifiable and visible to the community in the normal course of events) saying the wrong thing or not doing what they are meant to will attract penalties and bans.

It is not straightforward to justify immediate penalties for sins of omission — where someone fails to perform a duty — as opposed to actually doing and signing something wrong. That is because communications, power, etc, might cause interruptions that are not the agent's fault. For some functions an effective alternative is to ban agents from doing anything at all if they have missed a duty. How effective this is will mainly depend on whether the duty is still useful if performed late. This varies: of our own innovations hooks [26] are certainly useful even if moderately late, while releasing a contribution to a scheduled random oracle [29] is not. Indeed our designs are created to make — in large part — non contribution ineffective. Except for forcing the general population to do more work. The main motivation for the consensus machine idea introduced below is providing the required back-up mechanism: it allows us to initiate a decision on the assumption that (most of) the bad agents participate normally in the knowledge that the parameters will prevent a bad decision being made. (The back up allows a decision to be forced even when bad agents do not participate.)

In any case it will be clear that persuading potentially bad agents to do what is expected of good agents is an important part of getting dominated decisions, etc, made with moderately sized groups of pickets.

Note that these are formulated so it is stochastically impossible for bad nodes to have the system make a bad decision. A properly constructed group of pickets will result in either a clear and correct decision, which is what we want, or no decision.[5]

7.2 Deterministic decisions

It is most comfortable to engineer systems so that two good agents, if presented with the same inputs, will come to the same decision. That should be possible if they all base their calculations on exactly the same data, but may fail if they bring in private data such as their individual interpretations of time. Or it may fail if the execution involves sampling a large quantity of data. We would expect, however, that every decision arrived at by a good agent will be acceptable to (if not the first choice of) every good agent.

We cannot allow honest disagreement amongst good agents without factoring this out carefully in such a way as to allow a unique conclusion being reached. Of course this is more problematic in the pessimistic communication model where at the minimum we need $> 2/3$ of the agents to agree — which will impossible even for a binary choice when the good agents disagree. Consequently we

[5] In any multi-step agreement process like this where the subsequent calculations are based on earlier agreement, it is clearly possible to attempt agreement on multiple steps at the same time in anticipation that all parties turn out to agree on the earlier data. That would be more efficient if agreement is normal.

specify that all decisions must be based on the combination of public (to those deciding) data that is agreed by all plus decisions about the validity of candidate objects between which decisions are being made, and which contain all incomplete sampling. The latter can depend on both shared data (including in large quantities) and hidden data. (Our model for this is time or potentially other measured quantities.)

By the time a decision is actually made there needs to be agreement on whether the options are valid or not. The first author has previously suggested the traffic light model for this. In examining objects, each evaluator deems an object H to be one of

- Strictly good, or green.
- Liberally good, applying weaker criteria to local limits such as time. Amber. If any good agents deem green, all should deem at least amber.
- Bad, meaning a serious flaw found based on public data or serious divergence on local. Red. A red flaw should have readily checkable evidence that any agent can deterministically verify.

In this model the pickets or a wider population give a verdict. If there are any reds then (bearing in mind that there may be much evidence and two good agents may differ because one may find evidence that the other misses) then all pickets check this. With this evidence visible the pickets then decide whether H is

- Green: no valid reds and sufficient greens.
- Amber: no valid reds but insufficient greens.
- Red: at least one valid red.

H is valid just when green. Penalty regimes will differ for amber and red. In the above model, both the determination of validity and the final selection are separately subject to agreement.

8 Hierarchical consensus machines

We have already described how a consensus machine proceeds when it consists of a single set of pickets synchronising, in a rather abstract sense, through certificates appearing. We have, however, consciously envisaged ones that fail to make decisions, a situation very similar to the well-known phenomenon of deadlock. Deadlock is not normally an acceptable behaviour of a complete system, and certainly not in a blockchain that we expect to go on extending forever.

In this section we indicate how to recover from a deadlock in one machine by letting another take over. Specifically we show how control of a decision making procedure can be handed from one machine to another.

Any system running on a decentralised system that may have some misbehaving bad agents is bound to have complications when trying to understand it. Consensus machines are a mechanism for making such systems clear. Essentially

we try to create collections of agents from which the desired program becomes emergent behaviour, tolerating the presence of bad agents.

Our formulation is inspired by the large body of work on process algebra: understanding bodies of agents that run concurrently and interact by forms of synchronisation. In it, all the agents know the system description and the good agents do exactly what that requires of them. The synchronisations of this machine are certificates generated by the groups of pickets that are legitimately running, noting that we will ensure it impossible for any other to be started.

When passing decision making from one group of pickets to to another the transition might come because the first group has the evidence that it will not be able to decide, and passes the token across itself. Alternatively, almost certainly because bad agents fail to participate, it fails either to decide or see positively that it will not. In both cases we need to be careful that control will not be passed across when some agents in the first are already committed, or at least that then the second produces the same decision.

There is an interesting analogy here with process algebra. CSP [24, 27] has a number of ways in which one process can pass control to another. The throw operator `P [| A |> Q` runs like P until it throws an exception in the set A, which causes it to run like Q. On the other hand the interrupt operator `P /\ Q` has P run, but if Q performs any visible action it takes over.

The more difficult of these cases for us is where one group of pickets is taking over from another by its own action. That is because if a group of agents decides to hand over, there will be sufficient agreement to do so, excluding any other action. Handing over to a second group means that the first group has not made the big step decision and furthermore no member of the group can legitimately decide it has as that would be inconsistent with the decision that it cannot make one. The process taking over cannot have instantaneous effect on all the nodes of the other, so cannot eliminate all possibility of a decision emerging from there later. By watching events appear as certificates we cannot get the same clear effects that are usually assumed in process algebra.

In order to implement such handover operations properly, we need to design the participating machines properly and expose the necessary parts of their behaviour. This is of course a general principle. At the level of a unitary consensus machine the component agents need to make their signed outputs permanently visible to others. Each of them can have other internal machinations towards computing these. Of course any value or signal with other external significance needs to be exposed.

In implementing the handover protocol we end up with two or more unitary machines running side by side and interacting. We choose not to add any external framework but rather to modify the unitary machines so that their interaction is built in. The actions (i.e., certificates) of these modified machines fall into three categories:

1) The decisions that the hierarchical machine is making for the outside world, and others of interest outside this compound object. We expect here to promote these actions to ones of the combination.

2) Any decisions that are of interest to the others in the combination to guide their actions. In our protocol we abstract these interactions into writes into storage which are read by the others according to a nondeterministic discipline that we discuss below.
3) Ones that are only of internal interest to the individual unitary machines; typically representing synchronisations on having reached some point or agreement on data. These can be disregarded.

Given the interaction described under (2) using storage locations, we treat externally visible progress (1) as big steps and the others as small steps. We do not care about the pattern of small steps but will seek to ensure that everything visible from the outside implements the specification implied by high level models.

Suppose we have set up two machines G and H to make the same decision, where G is the primary mechanism and H is a back up, it is in some sense safe if each of the two is, and complete if we can rely that H eventually takes over if G does not give a complete resolution, and H itself being complete.

But this is not the whole story. When H starts this is because, most likely dues to a time out, it suspects that G is deadlocked. But G might just be slow, and furthermore signals from H may not prevent it and so a decision might emerge from each. Both would be fine in isolation, but the ambiguity of two might well be a disaster. For example if the decision is choosing the next block of a blockchain then this might lead directly to a fork, and so those attacking it might encourage the conditions for it to occur. In the next section we show how to prevent this possibility by implementing a protocol between G and H in the general terms set out above. We will specify and verify this in the next section. This protocol does not prevent H and G both issuing decisions, but makes sure that if they do they are the same.

With this we have the ability to create hierarchical consensus machines in which the control is first given to a safe but incomplete group of pickets that works efficiently in the event that not much of the enemy engages in overt misbehaviour or non-participation. They can hand over in one or several steps to less fragile mechanisms in the sense that it is harder to prevent a decision emerging. The bottom level will be a mechanism that, under standard assumptions, always deliver a correct decision but will typically involve more effort.

9 Modelling hierarchical consensus machines in CSP

There are two models presented below. The ideal model represents the behaviour of the decentralised G and H at an abstracted level. The G, H and storage locations used are not conventional sequential processes, but instead represent abstractions of the overall behaviour of many cooperating processes, configured to work together as consensus machines or equivalent. The model can therefore represent very general systems. The distributed model is an illustration of how this behaviour might be realised in terms of more concrete processes.

The protocol we present here has much in common with mutual exclusion. We want to prevent something akin to a race condition. An obvious question is whether we could use a simple mutex between G and H and only allow one to make the decision. The answer is no: it is part of the make-up of G that it can deadlock at any time. If it were to seek the right to make the decision but then deadlock, then the system would deadlock too — contrary to our specification.

9.1 Ideal model

Section temporarily redacted.

9.2 Distributed model

Section temporarily redacted.

10 Related work

Many classical protocols [23, 16, 15, 18] exist to solve the Byzantine agreement problem [22]. The emergence of blockchains renewed the research community's interest in this problem — and more generally on the problem of achieving consensus in distributed systems — leading to a number of new protocols [20, 5, 30, 32, 10, 13, 2, 4, 7].

The first consensus protocol proposed for blockchains was *Proof-of-Work* (PoW) in the context of Bitcoin [20]. Intuitively speaking, in this protocol, *miners* (i.e. block producer candidates) attempt to solve a cryptographic puzzle, and the first one who solves it is entitled to propose the next block to be added to the chain. Arguably, the main drawback of PoW protocols is how energy inefficient they can be [17, 31]; the larger the network the more computing power is used to constantly solve these cryptographic puzzles. *Proof-of-Stake* (PoS) protocols have been proposed [2, 10, 13, 6] as energy efficient alternatives to PoW ones. Hybrid PoW-PoS protocols have also been proposed [14].

In Proof-of-Stake protocols, agents signal their intention to participate in the block production process by *staking* a sum of cryptocurrency, i.e. the *stake*, they own. Staking means that this sum is locked (i.e. escrowed) for the duration of this process and it may be *slashed* as a means to punish bad behaviour. The frequency upon which agents are selected to participate in this process is proportional to the size of the stake. Note how in PoW computing power determines how often an agent is "selected" to produce a block as opposed to staked cryptocurrency in PoS. In PoS protocols, agents can be selected as a block producer but also as a member of a committee which is typically in charge of either electing block producers or *finalising blocks*, namely, determining whether a block is immutable and the only valid block at a given height. Before a block is deemed final, a number of candidate blocks at a given height might be "competing" to become final. Some PoS protocols rely on probabilistic mechanisms to determine

the finality of a block — e.g. Algorand [10], Ouroboros [13] — whereas some others rely on deterministic mechanisms — e.g. Internet Consensus Computer [7], Casper FFG [6], Tendermint [4]. Our handover protocol is meant to be used as a part of a PoS protocol to achieve deterministic finality. A PoS-based selection mechanism is used to choose committees of agents — their sizes are estimated based on our stochastic calculations — to implement machines G and H and to decide on the next *final* block using the handover protocol.

Despite being designed to be part of a fully-fledged blockchain consensus protocol, the handover protocol alone is closer in nature to Byzantine agreement protocols like [18, 21, 15, 32]. Abstractly speaking, these protocols have been designed around the use of the votes to form decisions and of a threshold/quorum to ensure safety. In fact, PBFT [18] specifically — and this voting mechanism more generally — has been a source of inspiration for many current blockchain protocols, including ours.

11 Conclusion

In this paper we have used formal tools to understand how consensus can arise in decentralised systems. Essentially we have set out a programmatic approach to laying down and analysing consensus: given a population of potential block creators and the potentially multiple perspectives of different users we need to establish a trust model that they are all happy with. We then have the job of having the blockchain select sufficient groups of pickets and decision criteria that all can be sure of any positive decisions they make.

On the assumption that we can incentivise most bad participants to participate apparently properly, this will give us all we need. But an essential part of such motivation is that the bad know that if they do not collaborate like this they will be defeated by a back up mechanism.

We have shown how to formalise both the primary and secondary mechanisms as Unitary Consensus Machines. While much of our treatment was inspired by process algebra, we were both able to design and verify the crucial protocol that links a hierarchy of decision making in CSP and FDR.

By allowing such hierarchical consensus decisions, we believe that we have tools for making blockchains more varied and flexible. We hope that our approach to creating the component machines which compose together to provide consensus can be automated.

It is only natural — to people steeped in such languages and tools — that CSP coupled with FDR is a good way to model complex interactions in decentralised consensus. We are pleased to have demonstrated the truth of this intuition. While the full systems representing consensus may be too involved to fit within the abstractions of such tools, it is comforting that like so many other areas of concurrent reasoning, we can find levels where they bring real benefit.

We hope that others will be found, and that our tools for bringing clarity to the topic of consensus will find many interesting applications.

Appendix 1: stochastic tables

These tables are based on the case where it is assumed that, to a good observer, each of a randomly selected group of agents has probability no more than p of behaving badly. This bad behaviour is generally assumed to be Byzantine: such agents can behave perfectly, do nothing at all from some point, or do absolutely anything else.

The randomised selection may be on the basis of head-count: each agent is equally likely to be selected. Or it may take account of some weighting such as the stake that each controls. For the latter, it works much better if the selection is with replacement: namely it is permitted for one agent to appear multiple times in the same group.

In Table 1, the left hand column shows the assumed probability p of choices being bad in the rows. The bottom row shows N the number of agents selected. Where there is a figure it is the smallest number of agents agreeing that stochastically proves, with a 10^{-18} threshold, that there is a good agent amongst them. Red entries in the top left are where even seeing all agents agreeing does not prove this, as it is deemed possible that all the agents are bad. In purple areas with an asterisk, the number is small and we clear of the coloured thresholds. The green and purple regions represent parameters where there are guaranteed to be more goods than bads, and respectively more than twice as many goods and bads. For example, with $p = 0.05$ and $N = 50$ we see that there can be no more than 16 bad choices, so 17 must (no matter how they are selected) will contain at least one good one, and all 50 will have at least 34 good. Thus this entry is in the purple zone where more than 2/3 are surely good. This what we need for constructing a decision making H that the bad agents cannot deadlock.

Recall that a strong dominated majority requires at least half the good nodes to agree. Wherever there is a number k in this table the largest possible number of bad nodes is one less than this figure and the smallest number of goods is $N - k + 1$. It follows that to have over half the good nodes agreeing we require $k + (N - k)/2$ to agree. Even if all the bad nodes are in this set, then those left are more than half the goods.

Table 2 supports situations where we want — to some degree of certainty — that less than half of the agents in a random sample are bad. This means, for example that by setting up a threshold encryption which is opened by exactly half

0.33333		40	47	53	65	76	127	219	304	467	
0.25		29	35	43	46	55	64	106	178	246	374
0.2	20	26	31	36	40	49	56	91	152	208	314
0.15	17	22	27	30	34	41	48	76	125	*	
0.1	14	18	21	24	27	33	37	59	*	*	*
0.05	9	12	15	17	19	22	*	*	*	*	
N	20	30	40	50	60	80	100	200	400	600	1000

Table 1. Calculations for at least one good agent.

		ϵ value (given as $log_{10}\epsilon$)					
		-3	-6	-9	-12	-15	-18
	M_σ	3.0902	4.76	6.002	7.03	7.943	8.759
p	0.3333	77	182	289	396	505	614
	0.25	29	68	109	149	190	231
	0.2	17	41	65	88	113	137
	0.15	10	24	38	52	66	80
	0.1	6	13	21	28	36	44
	0.05	3	6	9	12	15	18

Table 2. The table shows how many participants are needed to ensure that less than half are bad (with certainty $1 - \epsilon$), for each given likelihood p of any individual agent being bad. For each ϵ value, it also gives the equivalent multiple M_σ of the standard deviation σ, above the mean, that is exceeded with probability ϵ (see the appendix for further details).

the sample, the bad nodes involved can neither prevent the value being revealed by the good agents in the sample, nor can they prevent it being released. The table imagines that we may demand stochastic certainty for this or may tolerate a larger probability of failure, which would have to be allowed for.

The top row indicates, as a power of 10, the tolerated probability of failure ϵ. The second row translates this into a number of standard deviations in the normal distribution. The left hand column shows the assumed probability p of a bad agent, and the table entries show how large N has to be so that the likelihood of there being more than $N/2$ bad agents is less than the chosen ϵ.

References

1. A. W. Roscoe: Model-checking CSP, chap. 21. Prentice-Hall (1994), http://www.cs.ox.ac.uk/people/bill.roscoe/publications/50.ps
2. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Clark, J., Meiklejohn, S., Ryan, P.Y., Wallach, D., Brenner, M., Rohloff, K. (eds.) Financial Cryptography and Data Security. pp. 142–157. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
3. Brookes, S.D., Roscoe, A.: CSP: A Practical Process Algebra, p. 187–222. Association for Computing Machinery, New York, NY, USA, 1 edn. (2021), https://doi.org/10.1145/3477355.3477365
4. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. CoRR **abs/1807.04938** (2018), http://arxiv.org/abs/1807.04938
5. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform. https://ethereum.org/whitepaper/ (2014)
6. Buterin, V., Griffith, V.: Casper the friendly finality gadget. CoRR **abs/1710.09437** (2017), http://arxiv.org/abs/1710.09437

7. Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.A., Shoup, V., Williams, D.: Internet computer consensus. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. p. 81–91. PODC'22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3519270.3538430, https://doi.org/10.1145/3519270.3538430
8. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: TACAS. LNCS, vol. 8413, pp. 187–201 (2014)
9. Gibson-Robinson, T., Lowe, G.: Symmetry reduction in CSP model checking. International Journal on Software Tools for Technology Transfer **21**(5), 567–605 (2019)
10. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. p. 51–68. SOSP '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132747.3132757, https://doi.org/10.1145/3132747.3132757
11. Harel, D.: Statecharts: A visual formalism for complex systems. Science of computer programming **8**(3), 231–274 (1987)
12. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
13. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 357–388. Springer International Publishing, Cham (2017)
14. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August **19**(1) (2012)
15. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (may 1998). https://doi.org/10.1145/279227.279229, https://doi.org/10.1145/279227.279229
16. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (jul 1982). https://doi.org/10.1145/357172.357176, https://doi.org/10.1145/357172.357176
17. Li, X., Zhu, Q., Qi, N., Huang, J., Yuan, Y., Wang, F.Y.: Blockchain consensus algorithms: A survey. In: 2021 China Automation Congress (CAC). pp. 4053–4058 (2021). https://doi.org/10.1109/CAC53003.2021.9728000
18. Liskov, B.H., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. **16**(6), 1811–1841 (Nov 1994). https://doi.org/10.1145/197320.197383, https://doi.org/10.1145/197320.197383
19. Lowe, G.: Casper: A compiler for the analysis of security protocols. Journal of computer security **6**(1-2), 53–84 (1998)
20. Nakamoto, S., et al.: Bitcoin: a peer-to-peer electronic cash system (2008) (2008)
21. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference. p. 305–320. USENIX ATC'14, USENIX Association, USA (2014)
22. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (apr 1980). https://doi.org/10.1145/322186.322188, https://doi.org/10.1145/322186.322188
23. Rabin, M.O.: Randomized byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science (sfcs 1983). pp. 403–409 (1983). https://doi.org/10.1109/SFCS.1983.48
24. Roscoe, A.W.: The theory and practice of concurrency. Prentice Hall (1998)
25. Roscoe, A.W., Gardiner, P.H.B., Goldsmith, M.H., Hulance, J.R., Jackson, D.M., Scattergood, J.B.: Hierarchical compression for model-checking csp or how to check

1020 dining philosophers for deadlock. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 133–152. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
26. Roscoe, A.W., Lei, W.: Taking the work out of blockchain security. https://tbtl.com/wp-content/uploads/2020/09/Taking-the-work-out-of-blockchain-security.pdf (2020)
27. Roscoe, A.: Understanding Concurrent Systems. Springer (2010)
28. Roscoe, A., Antonino, P., Lawrence, J.: Embedding reverse links in a blockchain. In: Workshop Encouraging Building Better Blockchain Security (WEB3SEC'22) (2022), available at https://www.acsac.org/2022/workshops/web3sec/Roscoe2022.pdf
29. Roscoe, A., Chen, B.: Delay and escrow in the blockchain. https://tbtl.com/wp-content/uploads/2020/11/delayblock.pdf (2019), accessed: 2022-11-23
30. Wood, G.: Ethereum Yellow Paper. https://ethereum.github.io/yellowpaper/paper.pdf
31. Xiao, Y., Zhang, N., Lou, W., Hou, Y.T.: A survey of distributed consensus protocols for blockchain networks. IEEE Communications Surveys & Tutorials **22**(2), 1432–1465 (2020). https://doi.org/10.1109/COMST.2020.2969706
32. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. p. 347–356. PODC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3293611.3331591, https://doi.org/10.1145/3293611.3331591