Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Corso di Laurea in Informatica

MODELLIZZAZIONE E ANALISI DEL PROTOCOLLO DI ALGORAND

Relatore:
Chiar.mo Prof.
Cosimo Laneve
Correlatore:
Dott.ssa
Adele Veschetti

Presentata da:
Marco Madonna

III Sessione
Anno Accademico 2019/2020

Indice

Capitolo 1

Introduzione

La tecnologia emergente delle blockchain sta attirando sempre più attenzioni portando alla creazione di una grande quantità di sistemi che adottano questa tecnologia. Il veloce sviluppo che hanno avuto dalla creazione di Bitcoin sta portando alla luce i problemi di scalabilità, efficienza e velocità che l'approccio Proof of Work porta con sè. In questa tesi esamineremo Algorand [1], una blockchain recente, basata sulla Proof of Stake, che si propone di arginare queste problematiche, sviluppandone un modello e testandolo con il tool di analisi statistica PRISM [2] e verificandone sperimentalmente la resistenza alle fork e la rapidità nella creazione di nuovi blocchi, per poi modellizzare un possibile attacco tramite una partizione della rete e verificarne il comportamento anche in questo caso.

1.1 Struttura della tesi

Nel secondo capitolo verrà introdotto il concetto di blockchain e di algoritmo del consenso, inoltre verrà presentato Algorand e il suo protocollo.
Nel terzo capitolo verrà introdotto PRISM il tool per la quale è stato creato il modello e sulla quale sono stati eseguiti i vari test, con particolare attenzione verso il linguaggio che usa per la costruzione dei modelli e sull'estensione PRISM+ per l'analisi dei sistemi di blockchain.
Nel quarto capitolo vedremo come il modello di Algorand è stato costruito e i risultati dei test su esso condotti.
Infine nel quinto capitolo l'attenzione si sposterà sul modello di un possibile attacco ad Algorand ed il suo comportamento in caso si verifichi una partizione della rete.

Capitolo 2

Blockchain e algoritmi del consenso

2.1 Blockchain

La tecnologia blockchain implementa un registro dostribuito condiviso, decentralizzato e non alterabile. È definita come un *ledger*, letteralmente "libro mastro", e può essere considerato come il registro in cui si tiene traccia delle transazioni di un'attività.Una blockchain è costituita da una catena di blocchi, contenenti transazioni, in sequenza cronologica. I nodi che partecipano alla blockchain prendono parte ad una rete peer-to-peer e tengono in memoria una copia del ledger che aggiornano con nuovi blocchi partecipando al protocollo di consenso. Quest'ultimo serve a garantire la coerenza tra le varie copie del ledger e la validità dei nuovi blocchi creati. All'interno dei blocchi vengono salvati dati utili alle applicazioni che si basano sulla blockchain, come delle transazioni monetarie nel caso delle criptovalute, e dati utili al funzionamento della stessa: come un riferimento al blocco precedente, di solito ne viene salvato il valore hash, o valori utili all'utilizzo delle funzioni crittografiche utilizzate per creare nuovi blocchi o per gestire la partecipazione dei nodi al consenso. Una volta che un blocco è aggiunto al ledger non è più modificabile. Infatti se si modificasse un blocco si genererebbe un nuovo hash dello stesso producendo un effetto domino su tutti gli hash dei blocchi seguenti. Data la possibilità che si creino incosistenze tra i *ledger*, chiamate *fork*, un blocco per essere certamente incluso nella catena deve raggiungere una certa profondità partendo dall'ultimo blocco aggiunto, per Bitcoin ad esempio è 6, per Algorand vedremo che un blocco è certo fin da quando è aggiunto al ledger. Ogni blockchain gestisce il proprio algoritmo di consenso in maniera differente dalle altre.

2.2 Algoritmi del consenso

Non avendo un autorità centrale, i sistemi di blockchain, prima di aggiungere nuovi blocchi alla catena, devono far raggiungere un accordo ai diversi nodi che vi partecipano.

A questo scopo è di fondamentale importanza l'algoritmo di consenso utilizzato e le sue caratteristiche che ne desfiniscono proprietà come:

- l'efficienza, quanto tempo e risorse sono richieste ai nodi e alla rete per raggiungere il consenso
- l'affidabilità, quanto il sistema è prono a non incorrere in inconsistenze tra le varie copie del ledger
- la resistenza a possibili attacchi malevoli dall'esterno

Attualmente sono stati sviluppati diversi tipi di algoritmi del consenso, i più utilizzati rientrano in due principali categorie: Proof of Work (PoW) e Proof of Stake (PoS).
La Proof of Work è la prima famiglia di algoritmi inventati [3], è l'approccio adottato da Bitcoin e molte altre criptovalute. Caratteristica principale di questa famiglia di algoritmi è basare la scelta del nodo che propone il nuovo blocco sul processo di *mining*: il nodo con più potere computazionale ha più probabilità di riuscire a risolvere il criptopuzzle che lo abilita a creare il nuovo blocco. Questo approccio tende però ad accentrare il potere di scelta in poche grandi *pool*,grandi gruppi di nodi specializzati appartenenti ad un unica entità, di nodi e a venire meno al principio di decentralizzazione. Inoltre tutti i nodi per partecipare al consenso devono continuamemnte provare a minare nuovi blocchi portando ad un enorme dispendio di energia che può rendere la ricompensa ricevuta per la creazione del nuovo blocco irrilevante. Anche la sicurezza di questo protocollo è data dall' elevato potere computazionale e le vaste risorse richieste per corrompere la percentuale di rete necessaria a controllare la produzione di nuovi blocchi.
L'approccio Proof of Stake e alla base di Algorand perciò lo vedremo più del dettaglio qui di seguito.

2.2.1 Proof of Stake

La Proof of Stake, nata successivamente a quella PoW, propone un approccio alternativo basato sula quantità di token posseduti da un nodo piuttosto che sulla suo potere computazionale. Un token rappresenta una quantità di risorse distribuite dalla blockchain come ricompensa, nel caso di criptovalute rapresenta una quantità di moneta. Il sistema favorisce i nodi in base alla percentuale di token che posseggono sul totale di token presenti nel sistema, in base al principio secondo il quale un nodo con più token ha meno interesse a generare pericoli verso la struttura perché ha piu risorse da perdere. I nodi, quindi, non devono impiegare grandi risorse computazionali ed energetiche per partecipare, l'unica condizione per poter entrare a far parte della rete è possedere un account con almeno un token, slegato da risorse materiali, a suo nome. L'approccio PoS permette di velocizzare il processo di creazione di nuovi blocchi, riducendone allo stesso tempo il costo per chi partecipa, e permette agli *stakeholder*, i soggetti che posseggono un interesse, economico

e non, per quella blockchain, ad esempio un possessore di una determinata criptovaluta, di partecipare attivamente alla rete. Il sistema sarà sicuro finchè la maggioranza dei token apparterrà ad utenti onesti. I diversi algoritmi, che adottano l'approccio PoS, si diferenziano principalmente per l'effettivo metodo con la quale i nodi vengono scelti per prendere parte alla creazione dei nuovi blocchi e per le politiche adottate per gestire le stake dei partecipanti, con ricompense o penalità a seconda del comportamento dei nodi arrivando anche a cancellare completamente la stake in caso di comportamenti malevoli.

2.3 Algorand

Nel 2017 Silvio Micali fonda Algorand, una piattaforma digitale di pagamenti e valuta digitale, che si propone come alternativa sicura, scalabile e maggiormente distribuita rispetto ai metodo PoW maggiormente utilizzati nel campo delle criptovalute [4]. Algorand è una blockchain basata sulla PoS che utilizza un algoritmo incentrato sul problema dei generali Bizzantini per gestire un ledger pubblico senza un autorità centrale [5]. Il protocollo di consenso aggiorna il ledger selezionando casualmente dei comitati, formati dagli account collegati ai nodi della rete peer-to-peer, per proporre i blocchi ed effettuare i turni di voto per raggiungere il consenso. Per convalidare il blocco proposto in un turno bisogna raggiungere un limite minimo di voti, nello specifico il numero dei voti da raggiungere deve essere abbastanza da rendere sicuro il sistema anche nel caso ci siano utenti malevoli che intendono non far raggiungere un accordo. All'inizio di ogni round i partecipanti eseguono una funzione crittografica randomica per verificare se sono selezionati come propositori del nuovo blocco. Gli utenti che risultano scelti raccoglieranno le nuove transazioni dal sistema per costruire il nuovo blocco, una volta proposti i vari blocchi candidati iniziano i turni di voto per certificare il nuovo blocco da aggiungere, il blocco che avrà raggiunto il minimo di voti verrà aggiunto alla catena.La proposta di Algorand ha mostrato di poter raggingere il consenso su un nuovo blocco in relativamente poco tempo e di poter scalare facilmente su grandi numeri di utenti, in più [1] afferma che l'algoritmo PoS basato sul problema dei generali elimina la possibilità di inconsistenze con una probabilità irrisoria di generare fork.

2.3.1 Consenso in Algorand

Algorand utilizza un protocollo asincrono e distribuito organizzato in *round*, intervalli di tempo in cui un nodo svolge tutti i passaggi per certificare un nuovo blocco da aggiungere alla blockchain, il *round* r è adibito a costruire ed approvare il r-esimo blocco B^r e ogni nodo inizia il *round* r non appena riceve il blocco B^{r-1} certificato nel *round* in corso.
Ogni *round* è diviso in *period*, per ogni *period* un nodo esegue tutti gli *step* necessari per la proposta e la votazione del nuovo blocco, se alla fine di un periodo il nodo non ha ricevuto abbastanza voti per certificare un blocco passa al periodo successivo all'interno

dello stesso *round*. Il protocollo Algorand richiede che solo una parte di tutti i nodi che costituiscono la rete partecipino al consenso, all'inizio di ogni *step* un nodo verifica se fa parte del comitato scelto per partecipare.

Autodeterminazione dei comitati

Algorand costuisce il modo in cui ogni nodo decide se fa parte o meno del comitato partecipante per ogni step basandosi su una funzione verificabile e randomica (VRF) crittografica. La VRF genera un hash pseudocasuale e una prova pubblica della sua correttezza, applicando questa funzione un nodo sa se è selezionato e gli altri nodi conoscendo la prova e la chiave publica del nodo possono verificare l'effettiva legittimità della sua presenza nel comitato. L'input per la VRF è costituito da un seed pubbllicamente conosciuto salvato nella blockchain e dallo step per la quale sta avvenendo la selezione, proposta del blocco o uno degli step di voto. Il nodo inizia lo step calcolando la VRF con la sua chiave segreta e l'input descritto sopra, così facendo verifica se e quanta della sua stake entrerà in gioco in quella fase. Ogni token appartenente alla stake del nodo conta come un possibile membro del comitato, il numero di token selezionati dipende probabilità binomiale del numero di token previsti nel comitato sul totale, il numero previsto di token all'interno di un comitato dipende da qunti ne sono necessari per mantenere la sicurezza del sistema. Dalla prospettiva del sistema nel suo complesso questa selezione è spiegabile come n lanci di una moneta pesati, con n il numero di token nel sistema. Da parte del nodo si tratta della probabilità che k token appartenenti alla sua stake possano essere selezionati mentre c token sono previsti nel comitato da i S token presenti nel sistema. Secondo la distibuzione binomiale il numero di k token selezionati per un nodo che ha s token nella sua stake ha una probabilità esprimibile con la seguente formula [1] [6]:

$$B(k; s, p) = \binom{s}{k} p^k * (1-p)^{s-k}$$

Con:

- k il numero di token selezionati
- s la stake del nodo
- $p = c/S$ il rapporto tra il numero di token previsto nel comitato e il totale dei token presenti nel sistema

Più di un token può essere selzionato per partecipare al comitato, ogni token selezionato conta come un voto nelle fasi di voto.

Istruzioni del protocollo

Di seguito sono riportate le istruzioni per un generico nodo n. Ogni round r è formato da uno o più period, in qualunque momento un nodo lavora su esattamente un round e un period (r, p), un period è diviso in step (r, p, s) con s compreso tra 0 e 255. Per determinare quando agire in ogni step n usa un $timer_n$, il timer si resetta ogni volta che n inizia un nuovo period, i timer dei vari nodi non sono sinconizzati ma vanno alla stessa velocità, un secondo dura lo stesso tempo per tutti.
Per ogni step (r, p, s) viene eletto un comitato votante ed ogni step ha due parametri associati che dipendono esclusivamente da s:

- CS_s: il numero di token previsti nel comitato per lo step s
- CT_s: il quorum di voti richiesto nello step s

s	Step Name	CS	CT
0	propose	20	N/A
1	soft	2990	2267
2	cert	1550	1112
3...252	$next_{s-3}$	5000	3838
253	late	500	320
254	redo	2400	1768
255	down	6000	4560

Tabella 2.1: Valori dei parametri CS e CT per ogni step

Il nodo n inizia il round r con period 0 non appena vede un blocco certificato o un quorum di voti per la certificazione di un blocco del round $r - 1$. Mentre n è nel period (r, p) se riceve un next-quorum passa al period successivo.
Parametri temporali:

- λ: corrisponde al tempo richiesto per propagare un messaggio breve, ad esempio un voto, nella rete.
- Λ: corrisponde al tempo richiesto per propagare un messaggio lungo, ad esempio un blocco, nella rete.
- λ_f: la frequenza a cui gli step di fast recovery vengono ripetuti.

Di seguito descriviamo i vari step (r, p, s) del protocollo [6]:

- Step 0: [Proposal] $timer_n = 0$:

- se $p = 0$ o $p > 0$ e n ha ricevuto un next-quorum per $\perp$ nel period $(r, p-1)$, allora n costruisce una nuova proposta di blocco B_n, poi propaga B_n l'hash $H(B_n)$ come messaggi separati.
- Altrimenti se $p > 0$ e n ha ricevuto un next-quorum per $v = H(B) \neq \perp$ nel period $(r, p-1)$ allora n propaga v;

- Step 1: [Filtering] $timer_n = 2\lambda$:
  - se $p = 0$ o $p > 0$ e n ha ricevuto un next-quorum per $\perp$ del period $(r, p-1)$, allora n seleziona dalle proposte che ha ricevuto in questo period (se ne ha ricevute) la proposta v il cui propositore ha credenziali minime, ovvero la stringa hash delle credenziali che interpretata come un intero ha il valore minore, e soft-vota v;
  - Altrimenti se $p > 0$ e n ha ricevuto un next-quorum per $v = H(B') \neq \perp$ del period $(r, p-1)$,allora n soft-vota v.

- Step 2: [Certifying] $timer_n \in (2\lambda, max(4\lambda, \Lambda))$:
  - Se n riceve un soft-quorum di voti del period(r, p) per un valore v e un blocco valido B con $H(B) = v$, allora n cert-vota v.

- Step $s \in [3, 252]$: [Recovery] $timer_n = max(4\lambda, \Lambda)$ (quando $s = 3$) o $max(4\lambda, \Lambda) + 2s - 3\lambda + r$ (quando $4 \leq s \leq 252$), con $r \in [0, 2s - 3\lambda]$:
  - se n ha ricevuto un blocco valido B e un soft-quorum nel period (r, p) per $H(B)$ allora n next-vota $H(B)$;
  - Altrimenti se $p > 0$ e n ha ricevuto un next-quorum per $\perp$ nel period $(r, p-1)$, allora n next-vota $\perp$.
  - Altrimenti, se n ha ricevuto un next-quorum per $v = H(B') \neq \perp$ nel period $(r, p-1)$, allora n next-vota v.

- Step $s \in [253, 255]$: [Fast recovery] $timer_n = k\lambda_f + t$ con k intero positivo e $t \in [0, \lambda_f]$:
  - se n ha ricevuto un blocco valido B e un soft-quorum nel period (r, p) per $H(B)$ allora n late-vota $H(B)$ [step 253];
  - Altrimenti se $p > 0$ e n ha ricevuto un next-quorum per $\perp$ nel period $(r, p-1)$, allora n down-vota $\perp$ [step 255];
  - Altrimenti n ha ricevuto un next-quorum per $v = H(B') \neq \perp$ nel period $(r, p-1)$, allora n redo-vota v [step 254].

Gestione delle stake

In Algorand la criptovaluta collegata è l'Algo. La quantità massima nel sistema è stata fissata a 10.000.000.000 alla sua creazione. Le riserva di Algo da usare per le ricompense viene decisa ogni 500.000 round e conservata in un account utilizzato esclusivamente per pagare le ricompense, per questo la quantità di ricompense distribuita all'aggiunta di un nuovo blocco viene calcolata all'nizio di ogni periodo dividendo l'ammontare di Algo nell'account delle ricompense meno 0.1 algo, il minimo quantitativo di algo che un account può possedere, per il numero di round in cui le ricompense verranno distribuite (500.000). La riserva di Algo adibita alle ricompense viene decisa dalla fondazione Algorand, decidendo di conseguenza il "valore" di ogni singolo blocco aggiunto alla catena per ogni periodo di 500.000 round, ad esempio al momento della scrittura di questa tesi le ricompensa distribuita per ogni blocco e di 37.9 algo. L'ammontare della ricompensa ricevuta da ogni nodo, alla fine del round r, è proporzionale alla quantità di stake messa in gioco dal nodo al round r-322, in percentuale a quanto la sua stake contribuisce alla stake totale, e non è influenzata dal fatto che il nodo abbia partecipato o meno attivamente al consenso. Se la ricompensa spettante ad un nodo è minore di 1 microAlgo (10^{-6} algo) verrà accumulata nel prossimo blocco per esere elargita quando raggiungerà il minimo valore trasferibile di un microAlgo. L'ammontare di algo posseduto da un nodo viene aggiornato non appena l'account ad esso collegato eseguirà una transazione, sia in entrata che in uscita, registrata in un blocco [7].
Non sono previste penalità, la stake di un nodo diminuisce solo se l'account collegato effettua una transazione in uscita verso un altro account.

Capitolo 3

PRISM un tool di analisi probabilistica

Per analizzare le propietà di ALgotand, descritte alla fine del capitolo precedente, soprattutto la notevole efficienza nella prevenzione delle fork ripetto a Bitcoin e alle altre forme di consenso che si basano sulla proof of work, è stato scelto di utilizzare il tool PRISM, un model cheker probabilistico. In questa tesi ci concentreremo sui modelli CTMC (catene di markov tempo continue), sistemi di transizioni da uno stato s ad uno s' dove ogni transizione è regolata da un numero reale positivo ρ chiamato 'rate', la probabilità che avvenga una transizione da s ad s' in un tempo t è data dalla distribuzione esponenziale $1 - e^{-\rho t}$ [8]. Utilizzeremo questo tipo di modelli per rappresentare il protocollo distribuito di Algorand tenendo conto anche dei fattori probabilistici che entrano in gioco, dei tempi di attesa dei vari nodi e dei tempi si latenza della rete, come vedremo nel capitolo seguente. Una volta costruito il modello, PRISM permette di verificare sperimentalmente la probabilità che determinati eventi avvengano, come appunto la probabilità che si verifichi un' incosistenza tra le copie della blockchain salvate nei diversi nodi della rete.

3.1 PRISM

Per descrivere i sistemi da modellizzare, PRISM usa un linguaggio concorrente chiamato anch'esso PRISM. Un sistema è la composizione parallela di diversi moduli che possono interagire fra di loro, ogni modulo rappresenta un processo concorrente ed è composto dai suoi stati interni, rappresentati dai diversi valori che possono assumere le variabili appartenenti al modulo, e dall'insieme dei suoi comandi che ne determinano il comportamento. I comandi stabiliscono sotto quali condizioni può avvenire una transizione e come essa andrà a modificare lo stato del modulo attrverso la modifica delle variabili interne. I comandi sono scritti nella forma :

```
[label] guard -> r_1:update_1+...+rate_n:update_n;
```

dove:

- `label` è un etichetta che serve a sincronizzare moduli diversi: i comandi con la stessa etichetta appartenenti a moduli diversi verranno eseguiti insieme, nel caso in cui la label è vuota il comando non si sincronizza con nessun altro.
- `guard` è un predicato su qualsiasi variabile presente nel sistema, anche non appartenente al modulo del quale il comando fa parte.
- `r_i` è il rate della rispettiva transizione.
- `update_i` è una transazione definita come asegnazione di un nuovo valore ad una variabile del modulo, nella forma:

```
(v_1' = exp_1)&...&(v_n' = exp_n)
```

dove:

- `v_i` è il nome di una variabile appartenente al modulo
- `exp_i` è un epressione il cui valore sarà assegnato alla variabile `v_i`.

Quando la guardia risulta vera verrà eseguita una transizione, associata ad essa, scelta in base al rate `r_i` attribuito a ciascuna, tipicamente da uno stato del sistema si possono raggiungere più di uno stato e il nuovo stato sarà deciso dalla prima transizione eseguita [9]. Un modulo sarà quindi scritto secondo la forma

```
module M
    //variabili
    v : [0..N];
    v1 : int init 0;
    b : bool init false;
    //altre possibili variabili...

    //comandi
    [] (v > 0) -> r : (v1' = v1 +1);
    [label] (v = 0) -> r1 : (b' = true);
    //altri possibili comandi...
endmodule
```

dove:
`module` e `endmodule` indicano rispettivamente l'inizio e la fine del modulo di nome `M`
ed al suo interno possiamo vedere degli esempi di dichiarazioni di variabili, nello specifico:
`v` è una varibile intera di range `0...N`
`v1` è una variabile intera senza range inizializzata a `0`
`b` è una variabile booleana inizializzata a `false`
e due esempi di comandi di cui il secondo sincronizzato con un comando di un altro modulo tramite l'etichetta `label`.

3.2 PRISM+

Il linguaggio base PRISM non è tuttavia sufficiente per modellizzare il protocollo di consenso di una blockchain. Per rappresentare fedelmente il protocollo è necessario utilizzare i tipi di blocco e di ledger, per usufruire di questi utilizzeremo PRISM+ un estensione di PRISM che introduce i tipi che ci servono. PRISM+ è un estensione creata per modellizzare il protocollo Bitcoin per cui oltre al tipo blocco e ledger implementa anche il tipo coda per simulare i blocchi salvati nel miner ma non ancora inseribili nel ledger, che nel nostro modello non utilizzeremo. Di seguito il link alla repository di PRISM+: `https://github.com/adeleveschetti/bitcoin-analysis`

3.2.1 Blocco

Il tipo dati "blocco" è una versione semplificata del blocco utilizzato da Bitcoin: vengono esclusi dal modello tutti quei dati, come le transazioni e i valori hash del blocco, che pur essendo utili al funzionamento della bockchain non prendono parte al processo del consenso. Al netto di questi dati interni al blocco, in Algorand abbiamo, oltre alle transazioni salvate, anche il bilancio degli utenti e altri valori che servono come input alle funzioni hash utilizzate che vedremo nel capitolo successivo, è possibile utilizzare il blocco di PRISM+ per rappresentare anche i blocchi usati in algorand. Nello specifico un blocco è una coppia ordinata di nomi `{name,father}` dove il primo è il nome che identifica il blocco e il secondo, `father`, è l'identificativo del blocco precedente al quale è connesso. Un nome è anch'esso una coppia di valori `m,c`: `m` è l'identificativo del modulo che ha creato il blocco e `c` nel nostro modello è il numero del round in cui il blocco è stato creato, in Bitcoin è il c-esimo blocco che è stato minato dal miner m. Un esempio di blocco che potremmo trovare nel nostro modello è `{m2,3;m1,2}` che denota il blocco creato a round `3` dal modulo `m2` che è preceduto dal blocco creato nel round `2` dal modulo `m1` [9].

3.2.2 Ledger

Il tipo ledger è una coppia `<T;p>`, `T` è un albero di blocchi dove ogni blocco punta al proprio blocco padre e `p` è un puntatore alla foglia di profondità massima, nella pratica il nome che identifica l'ultimo blocco. La radice dell'albero è il blocco genesi `{genesis,0;genesis,0}`. Nel nostro caso ogni modulo aggiunge un solo blocco per round al suo ledger, per cui quest'ultimo sarà più assimilabile ad una lista e quindi all'effettiva blockchain. Utilizzeremo le seguenti operazioni legate a questi tipi:

- `addBlock(ledger,block)`: dove il primo argomento è un ledger e il secondo è un blocco, l'operazione aggiunge il blocco al ledger, se possibile, ovvero se il padre del blocco è presente nel ledger.
- `createBlock(ledger,integer)`: dove il primo argomento e un ledger ed il secondo è un intero c che nel modello rappresenta il numero del round, verrà quindi creato un blocco che ha per padre l'ultimo blocco nel ledger e per nome la coppia nome del modulo,c.

Capitolo 4

Modellizzazione e analisi di Algorand

4.1 Modello di Algorand in PRISM+

Questa sezione è dedicata alla spigazione del modello PRISM+ di Algorand e a come è stato costruito basandosi principalmente sul più recente articolo di presentazione [1] e sul documento di specifiche dell'Algorand foundation [6] oltre che su varie fonti di altri modelli fatti [10] [11].
Il modello è costituito da 8 nodi rappresentati da altrettanti moduli (denominati M,M1...M7) e un modulo chiamato Global che rappresenta la rete e tiene conto dei messaggi scambiati dai nodi: voti e blocchi. Inoltre sono presenti due moduli aggiuntivi StakeManager e ForkControll, il primo si occupa della gestione delle ricompense per i vari nodi ed il secondo controlla la presenza di incosistenze tra i ledger dei vari nodi.

4.1.1 I Moduli M

I moduli M rappresentano i nodi della rete e sono identici fatta eccezione per i nomi delle variabili che variano da modulo a modulo.

```
1  module M
2
3    round : int init 0; //round number
4    p : int init 0; //period number
5          s : [0..15] init Proposal; //step number
6
7    B : blockchain [{genesis,0;genesis,0}];// copy of the blockchain
       saved in the node
8    b : block {m,0;genesis,0};//block the node is working on
9    c : int init 0;
10   blockReceived : bool init false;
```

```
11
12
13  //propose step
14  [] (s = Proposal) -> (1- (pow(1-(pcs/totStake),stake) )) * sw : (s' =
      Proposer) +
15              pow(1-(pcs/totStake),stake) * sw : (s' = Filtering);
16
17  [] ((s = Proposer)&(p = 0)) | ((s = Proposer)&(p > 0)&(nextNilVotes0
     >= nextQuorum))  -> 1 : (b' = B + c)&(s' = Propose);
18  [] (s = Proposer)&(p > 0)&(nextVotes0 >= nextQuorum) -> 1 : (b' =
     prevBlock)&(s' = Propose);
19
20  [proposeBlock] (s = Propose) -> sw : (s' = Filtering);
21
22
23  //filtering step
24  [] (s = Filtering) -> (1- (pow(1-(fcs/totStake),stake) ))  * sw : (s'
      = Filterer) +
25                pow(1-(fcs/totStake),stake) * sw : (s' = Update);
26
27  [] ((s = Filterer)&(p = 0)) | ((s = Filterer)&(p > 0)&(nextNilVotes0
     >= nextQuorum)) -> 1 : (b' = actBlock)&(blockReceived' = true)&(s' =
      SoftVote);
28  [] ((s = Filterer)&(p > 0)&(nextVotes0 >= nextQuorum)) -> 1 : (b' =
     prevBlock)&(blockReceived' = true)&(s' = SoftVote);
29
30  [softVote] (s = SoftVote) -> lw : (s' = Certifying);
31
32  [] ((s = Update)&(p = 0)) | ((s = Update)&(p > 0)&(nextNilVotes0 >=
     nextQuorum)) -> lw : (b' = actBlock)&(blockReceived' = true)&(s' =
     Certifying);
33  [] ((s = Update)&(p > 0)&(nextVotes0 >= nextQuorum)) -> lw : (b' =
     prevBlock)&(blockReceived' = true)&(s' = Certifying);
34
35
36  //certifyng step
37  [delNexVotes] (s = Certifying) -> (1- (pow(1-(ccs/totStake),stake) ))
      * sw : (s' = Certifier) +
38                                  pow(1-(ccs/totStake),stake) * sw : (s'
      = Certificate);
39
40  [certVote] (s = Certifier)&(softVotes0 >= softQuorum) -> sw : (s' =
     Certificate);
41  [] (s = Certifier)&(softVotes0 < softQuorum) -> sw : (s' = Recovery);
42
43  [certifyBlock] (s = Certificate)&(certVotes0 >= certQuorum) -> 1 : (s
     ' = AddBlock);
44  [] (s = Certificate)&(certVotes0 < certQuorum) -> sw : (s' = Recovery
     );
```

```
45
46
47   //ending round step
48   [addBlock] (cert = true) -> 1 : (b' = certBlock)&(s' = AddBlock);
49   [] (s = AddBlock) -> 1 : (B' = B + b)&(s' = EndRound);
50   [endRound] (s = EndRound) -> 1 : (round' = round+1)&(c' = c +1)&(p' =
       0)&(blockReceived' = false)&(s' = Proposal);
51
52
53   //recovery step
54   [] (s = Recovery) -> (1- (pow(1-(rcs/totStake),stake) )) * sw: (s' =
      Recoverer) +
55                                pow(1-(rcs/totStake),stake) * sw: (s'=
      NextVotesCheck);
56
57   [nextVote] (s = Recoverer)&(blockReceived = true)&(softVotes0 >=
      softQuorum) -> 1 : (s' = NextVotesCheck);
58   [nextNilVote] (s = Recoverer)&((blockReceived = false) | (softVotes0
      < softQuorum)) -> 1 : (s' = NextVotesCheck);
59
60   [] (s = NextVotesCheck)&((nextVotes0 >= nextQuorum) | (nextNilVotes0
      >= nextQuorum)) -> 1 : (s' = EndPeriod);
61   [] (s = NextVotesCheck)&(nextVotes0 < nextQuorum)&(nextNilVotes0 <
      nextQuorum) -> 1 : (s' = Recovery);
62
63
64   //ending period step
65   [] (s = EndPeriod) -> 1 : (blockReceived' = false)&(p' = p+1)&(s' =
      Proposal);
66
67
68 endmodule
```

Listing 4.1: Codice del modulo M

Variabili

Dalla riga 0 alla riga 10 possiamo vedere le variabili assegnate ai moduli che simulano i nodi della rete. Partendo dall'alto abbiamo le prime tre variabili `round`, `p` ed `s` in cui il modulo tiene traccia del rispettivamente del round, period e step nella quale si trova. Subito dopo abbiamo la variabile `B` di tipo blockchain che rappresenta la copia del ledger salvata nel nodo che verrà aggiornata ogni round con il nuovo blocco creato. Dopodiché c'è `b` una variabile di tipo block nella quale verrà salvato il blocco proposto o il blocco al momento esaminato per le votazioni. Infine abbiamo due varibili di supporto `c` e `blockReceived` la prima è un contatore che tiene conto del numero del round e servirà al nodo per creare il blocco da proporre e la seconda di tipo booleano verrà settata a `true` una volta che il nodo avrà ricevuto il blocco sulla quale lavorerà per quel period.

Step di proposta

Il primo step è descritto dalla riga 13 alla riga 20. Il modulo parte dal round 0 period 0 e dallo step Proposal, step in cui si seleziona i nodi che parteciperanno al comitato di proposta del blocco. Come visto precedentemente la probabilita che k token vengano scelti per far parte di un comitato è:

$$B(k; s, p) = \binom{s}{k} p^k * (1-p)^{s-k}$$

Sviluppando la formula con $k = 0$ otteniamo la probabilità che 0 token vengano selezionati, quindi la probabilià che il nodo non faccia parte del comitato :

$$P_0 = (1-p)^s = (1 - \frac{c}{S})^s$$

dove:

- c è il numero di token previsti nel comitato per questo step, in questo caso 20
- S è la quantità totale di stake nel sistema
- s è la quantità di stake posseduta dal nodo

Essendo P_0 la probabilità che un nodo non venga scelto $1 - P_0$ è la probabilità che un nodo venga scelto per far parte del comitato. Queste due formule moltiplicate per il rate derivante dal tempo che il nodo aspetta per eseguire questo passaggio le troviamo come rate del primo comando, che appunto serve a modellizzare l'autoselezione che ogni nodo esegue all'inizio di ogni step e che rivedremo analogamente all'inizio degli step successivi. Se il nodo è selezionato entrera nello stato di `Proposer` dove dopo aver eseguito i controlli sui next-votes ricevuti nel period precedente, come visto nella descrizzione del protocollo Algorand al capitolo 2, deciderà se costruire un nuovo blocco, con l'istruzione `(b' = B + c)`, o riproporre il blocco del period precedente, passando quindi allo stato di `Propose` dove si sincronizzerà con il modulo Global per proporre il blocco agli altri nodi. Se il nodo non viene selezionato passerà direttamente allo step successivo.

Step di pre-voto

Nel secondo step (righe dalla 23 alla 33), come nel precedente, dopo aver cotrollato se il nodo entra a far parte o meno del comitato, in entrambi i casi dopo aver effetuato i necessari controlli sui next-voti del period precedente il nodo aggiorna il blocco su cui sta lavorando in base a quello scelto dal modulo Global. Questo passo che nel vero protocollo viene eseguito da ogni nodo che decide da se il blocco proposto dal nodo con l'hash delle

credenziali minore, ma che nel modello viene scelto dal nodo Global in quanto il risultato della funzione hash è randomico ed effettuando la scelta in ogni nodo non si avrebbe un uniformità tra tutti i moduli. Se il nodo viene selezionato si sincronizzerà con il modulo Global per inviare i suoi soft-votes alla rete, altrimenti passerà direttamente allo step successivo di certificazione.

Step di certificazione

In questo step (righe dalla 36 alla 38) adibito alla certificazione del blocco, se il nodo entra a far parte del comitato controllera i soft-voti ricevuti dal blocco in esame, se raggiungono il quorum passerà allo stato Certificate, dove controllerà i cert-voti e se sono abbastanza si sincronizzerà con il modulo Global per passare agli altri il blocco certificato. Se invece non partecipa al comitato passerà direttamente allo stato Certificate e proverà a certificare il blocco. Se in uno di questi stati i voti non dovessero esseri sufficienti si passa allo step di Recovery. Una volta che un nodo ha cerificato il blocco il modulo Global si sincronizzerà con tutti i nodi per passare il blocco ed ogni nodo una volta ricevuto il blocco lo aggiungerà al suo ledger, terminando poi il round e aggiornando le proprie variabili di conseguenza (righe dalla 47 alla 50).

Step di recupero

Nel caso in cui un nodo non dovesse ricevere i voti necessari a certificare un blocco, e nemmeno un blocco certificato dal modulo Global, entrerà nello step di Recovery (righe dalla 53 alla 61) dove rimarrà fino a che non riceverà un quorum di next-voti per un blocco o per per nessun blocco, continuando a votare se selezionato nel comitato altrimenti verificando solo i voti ad ogni ciclo. Una volta raggiunto il quorum terminerà il period e passerà a quello successivo aggiornando le variabili di conseguenza (righa 65). Gli ultimi step di Fast Recovery non sono stati modellizzati in quanto servono per ripristinare la rete in casi di emergenza, ma i voti espressi in questa fase non entrano in gioco durante la procedura di consenso per certificare il blocco.

4.1.2 Il modulo Global

Il modulo Global simula lo scambio di voti e blocchi tra i vari nodi della rete, scambio che nel protocollo reale avviene tramite gossip: ogni nodo comunica i suoi messaggi ai nodi vicini che a loro volta li passerano a quelli a loro adiacenti. In esso sono salvati i voti recapitati ad ogni nodo, i blocchi proposti, il blocco selzionato e quello del period precedente come si può vedere nel codice di seguito.Viene riportato solo il codice relativo alle variabili comuni a tutti i nodi e quello relativo alle varibili del modulo M, le altre variabili dei moduli saranno uguali solo con il numero del modulo al posto dello 0.

```
1 module Global
2
3     prevBlock : block {e,0;gesesis,0};
4     actBlock : block {e,0;gesesis,0};
5     propCount : int init 0;
6     prop : bool init false;
7     certBlock : block {e,0;gensis,0};
8     cert : bool init false;
9
10    block0 : block {e,0;genesis,0};
11    block0rec : bool init false;
12    softVotes0 : int init 0;
13    certVotes0 : int init 0;
14    nextVotes0 : int init 0;
15    nextNilVotes0 : int init 0;
```

Listing 4.2: Variabili del modulo Global

Le variabili `block`(righe 3 e 4 del listing 4.2) della prima parte salvano i blocchi sulla quale la rete sta lavorando, dopo aver selezionato tra le varie proposte.La varibile `propCount` conta il numero di proposte ricevute, `prop` è un booleano che indica quando è stato selezionato un blocco per il period, `certBlock` salva il blocco una volta che è stato certificato per passarlo poi ai nodi e `cert` è un booleano che indica quando un blocco è stato certificato. Nella seconda parte viene salvato il blocco proposto dal relativo nodo, il booleano `block0rec` indica che si è ricevuto un blocco proposto dal nodo, le restanti variabili servono a contare i voti ricevuti dal relativo nodo. Di seguito i comandi del modulo Global che si sincronizzano con i vari nodi quando eseguono azioni di voto, proposta o certificzione dei blocchi.

```
1 [proposeBlock] (s = Propose)   ->
2 1 : (block0' = b)&(block0rec' = true)&(propCount' = propCount + stake);
3 [proposeBlock1] (s1 = Propose) ->
4 1 : (block1' = b1)&(block1rec' = true)&(propCount' = propCount + stake1
    );
5
6 [] (propCount >= pcs)&(block0rec = true)&(prop = false) ->
7 (prevBlock' = actBlock)&(actBlock' = block0)&(prop' = true);
8 [] (propCount >= pcs)&(block1rec = true)&(prop = false)  ->
9 (prevBlock' = actBlock)&(actBlock' = block1)&(prop' = true);
```

Listing 4.3: Comandi di Global per la proposta dei nuovi blocchi

Nel listing 4.3 abbiamo le operazioni di proposta e selezione del blocco per il modulo `M` e `M1`, nel modello ce n'è una per ogni nodo. Quando un nodo propone un blocco qusto viene salvato nella variabile `block` ad esso collegata, viene aggiornato il contatore dei voti per le proposte e la variabile che indica che è stato ricevuto un blocco da quel nodo (righe da 1 a 4). Una volta che si è raggiunto il numero di proposte aspettato il modulo effettuerà una scelta non deterministica tramite i comandi della seconda parte (righe da

6 a 9), le proposte che arriveranno dopo che la scelta del blocco che verrà salvato in `actBlock`, non verranno prese in considerazione fino al termine del period.

```
[softVote] (s = SoftVote) ->
1 : (softVotes0' = softVotes0 + stake)&(softVotes1' = softVotes1 +
    stake)&
(softVotes2' = softVotes2 + stake)&(softVotes3' = softVotes3 + stake)&
(softVotes4' = softVotes4 + stake)&(softVotes5' = softVotes5 + stake)&
(softVotes6' = softVotes6 + stake)&(softVotes7' = softVotes7 + stake);

[certVote] (s = Certifier) ->
1 : (certVotes0' = certVotes0 + stake)&(certVotes1' = certVotes1 +
    stake)&
(certVotes2' = certVotes2 + stake)&(certVotes3' = certVotes3 + stake)&
(certVotes4' = certVotes4 + stake)&(certVotes5' = certVotes5 + stake)&
(certVotes6' = certVotes6 + stake)&(certVotes7' = certVotes7 + stake);

[nextVote] (s = Recoverer) ->
1 : (nextVotes0' = nextVotes0 + stake)&(nextVotes1' = nextVotes1 +
    stake)&
(nextVotes2' = nextVotes2 + stake)&(nextVotes3' = nextVotes3 + stake)&
(nextVotes4' = nextVotes4 + stake)&(nextVotes5' = nextVotes5 + stake)&
(nextVotes6' = nextVotes6 + stake) &(nextVotes7' = nextVotes7 + stake);

[nextNilVote] (s = Recoverer) ->
1 : (nextNilVotes0' = nextNilVotes0 + stake)&
(nextNilVotes1' = nextNilVotes1 + stake)&
(nextNilVotes2' = nextNilVotes2 + stake)&
(nextNilVotes3' = nextNilVotes3 + stake)&
(nextNilVotes4' = nextNilVotes4 + stake)&
(nextNilVotes5' = nextNilVotes5 + stake)&
(nextNilVotes6' = nextNilVotes6 + stake)&
(nextNilVotes7' = nextNilVotes7 + stake);
```

Listing 4.4: Comandi di Global per le operazioni di voto

I comandi dedicati alle operazioni di voto semplicemente aggiornano il conteggio dei voti per ogni blocco(listing 4.4). Ogni modulo avrà il suo corrispettivo comando per aggiornare i voti di tutti gli altri.

```
[certifyBlock] (s = Certificate) ->
1 : (certBlock' = b)&(cert' = true);

[addBlock] (cert = true) ->
1 : (cert' = false)&(prop' = false)&(propCount' = 0)&
(block0rec' = false)&(block1rec' = false)&(block2rec' = false)&
(block3rec' = false)&(block4rec' = false)&(block5rec' = false)&
(block6rec' = false)&(block7rec' = false);
```

Listing 4.5: Comandi di Global per la certificazione del nuvo blocco

Quando un blocco è certificato da un nodo viene salvato in `certBlock` e il flag `cert` viene settato a `true` per attivare il comando `[addBlock]` che sincronizzandosi con tutti i nodi gli passa il blocco che è appena stao certificato(listing 4.5).

```
1 [delNexVotes] (s = Certifying) -> 1 : (nextVotes0' = 0)&(nextNilVotes0'
      = 0);
2
3 [endRound] (s = EndRound) -> 1 : (softVotes0' = 0)&(certVotes0' = 0);
4
5 [endPeriod] (s = EndPeriod) -> 1 : (softVotes0' = 0)&(certVotes0' = 0);
```

Listing 4.6: Comandi di Global per l'azzeramento del conteggio dei voti

Questi comandi servono ad azzerare i voti salvati in Global per ogni nodo quando questi cambia round o period(listing 4.6).

4.1.3 I moduli StakeManager e ForkControll

```
1 module StakeManager
2
3   totStake : int init 8000; //total stake in the system
4
5   stake : int init 1000;
6   stake1 : int init 1000;
7   stake2 : int init 1000;
8   stake3 : int init 1000;
9   stake4 : int init 1000;
10  stake5 : int init 1000;
11  stake6 : int init 1000;
12  stake7 : int init 1000;
13
14  [addBlock] (cert = true) -> 1 : (totStake' = totStake + floor(
      rewardPerRound));
15
16  [endRound] (s = EndRound) -> 1 : (stake' = stake + floor((stake/
      totStake)*rewardPerRound));
17  [endRound1] (s1 = EndRound) -> 1 : (stake1' = stake1 + floor((stake1/
      totStake)*rewardPerRound));
18  [endRound2] (s2 = EndRound) -> 1 : (stake2' = stake2 + floor((stake2/
      totStake)*rewardPerRound));
19  [endRound3] (s3 = EndRound) -> 1 : (stake3' = stake3 + floor((stake3/
      totStake)*rewardPerRound));
20  [endRound4] (s4 = EndRound) -> 1 : (stake4' = stake4 + floor((stake4/
      totStake)*rewardPerRound));
21        [endRound5] (s5 = EndRound) -> 1 : (stake5' = stake5 + floor((
      stake5/totStake)*rewardPerRound));
22  [endRound6] (s6 = EndRound) -> 1 : (stake6' = stake6 + floor((stake6/
      totStake)*rewardPerRound));
```

```
23   [endRound7] (s7 = EndRound) -> 1 : (stake7' = stake7 + floor((stake7/
      totStake)*rewardPerRound));
24
25 endmodule
```

Listing 4.7: Codice del modulo StakeManager

Il modulo `StakeManager` (listing 4.7) gestisce la stake dei nodi, secondo le politiche di Algorand, aggiornando ad ogni fine round la stake dei nodi dividendo il premio del round fra tutti i nodi in percentuale alla loro stake e aggiornando la stake globale di conseguenza.
Il modulo `ForkControll` (listing 4.8) si occupa di controllare alla fine di ogni round le inconsistenze tra i vari ledger salvati nei blocchi, utile per le analisi statistiche che vedremo nel prossimo paragrafo.

```
 1 module ForkControll
 2
 3   diff : int init 0;
 4   diff1 : int init 0;
 5   diff2 : int init 0;
 6   diff3 : int init 0;
 7   diff4 : int init 0;
 8   diff5 : int init 0;
 9   diff6 : int init 0;
10   diff7 : int init 0;
11
12
13   [endRound] (s = EndRound) -> 1 : (diff' = max(B-B1,B-B2,B-B3,B-B4,B-
      B5,B-B6,B-B7));
14   [endRound1] (s1 = EndRound) -> 1 : (diff1' = max(B1-B,B1-B2,B1-B3,B1-
      B4,B1-B5,B1-B6,B1-B7));
15   [endRound2] (s2 = EndRound) -> 1 : (diff2' = max(B2-B1,B2-B,B2-B3,B2-
      B4,B2-B5,B2-B6,B2-B7));
16   [endRound3] (s3 = EndRound) -> 1 : (diff3' = max(B3-B1,B3-B2,B3-B,B3-
      B4,B3-B5,B3-B6,B3-B7));
17   [endRound4] (s4 = EndRound) -> 1 : (diff4' = max(B4-B1,B4-B2,B4-B3,B4
      -B,B4-B5,B4-B6,B4-B7));
18   [endRound5] (s5 = EndRound) -> 1 : (diff5' = max(B5-B1,B5-B2,B5-B3,B5
      -B4,B5-B,B5-B6,B5-B7));
19   [endRound6] (s6 = EndRound) -> 1 : (diff6' = max(B6-B1,B6-B2,B6-B3,B6
      -B4,B6-B5,B6-B,B6-B7));
20   [endRound7] (s7 = EndRound) -> 1 : (diff7' = max(B7-B1,B7-B2,B7-B3,B7
      -B4,B7-B5,B7-B6,B7-B));
21
22 endmodule
```

Listing 4.8: Codice del modulo ForkControll

4.2 Analisi dei dati ottenuti

In questo paragrafo verranno illustrati i dati ricavati dalle simulazioni effettuate sul modello precedentemente descritto. Mi concentreremò sui principali vantaggi che Algorand può avere rispetto all'approccio Proof of Work più classico, ad esempio quello di Bitcoin, ovvero:

- L'efficienza: si riesce a creare un nuovo blocco in in meno tempo e con l'utilizzo di una minor quantità di risorse.
- L'affidabilità: la probabilità che si verifchi una fork è considerevolmente inferiore

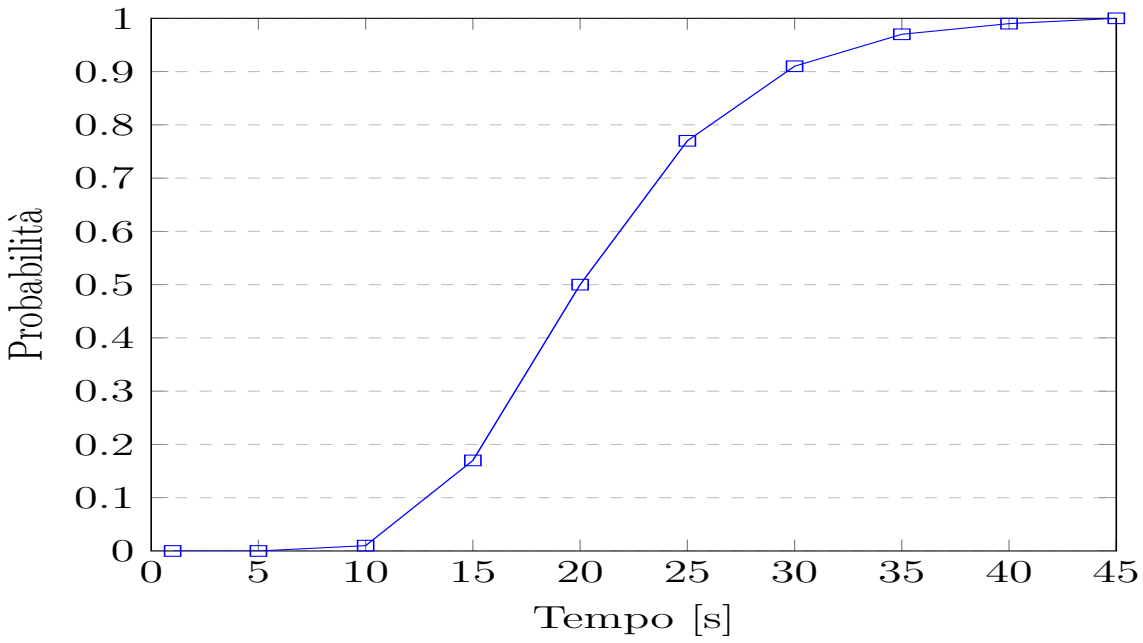

Figura 4.1: Probabilità in Algorand che un nuovo blocco venga creato in tempo T

Nella Figura 4.1 possiamo notare come dopo 45 secondi la probabilità che un nuovo blocco venga creato è uguale ad 1 e il tempo medio per la produzione di un blocco si attesta intorno ai 20 secondi. Il tempo di creazione di un nuovo blocco è molto minore rispetto a quello dell'approccio PoW, in Bitcoin ad esempio il tempo di creazione di un blocco è fissato a 10 minuti, come si può vedere nella Figura 4.2 estratta dall'articolo [9]. In questo modo le transazioni possono essere salvate più rapidamente nella blockchain ottenendo così una maggiore velocità nello scambio della criptovaluta.

Nella Figura 4.3 vediamo la probabilità che si verifichi una inconsistenza di un blocco tra le varie copie del ledger. La probabilità si attesta nell'ordine del 10^{-4},nonostante aumenti in base al tempo poiché in più tempo vengono creati più blocchi e quindi abbiamo

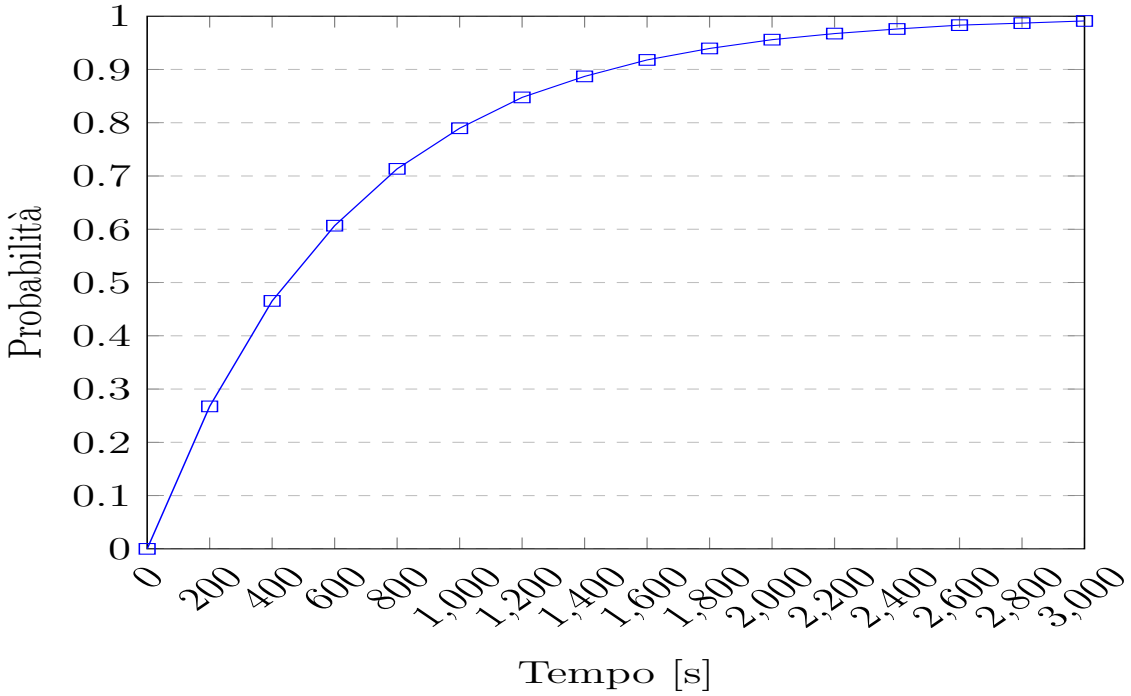

Figura 4.2: Probabilità in Bitcoin che un nuovo blocco venga creato in tempo T

più occasioni che si verifichi una fork. Questa probabilità è di due ordini di grandezza inferiore rispetto a quella riscontrata nel modello Bitcoin come si può vrdere in Figura 4.4, nonostante in 600s nel caso di Bitcoin venga creato un solo blocco mentre nel modello di Algorand vengano creati circa 30 blocchi. Come vediamo anche nella Figura 4.5 dove è riportata la probabilità che si verifichi una fork di lunghezza crescente con tempo fissato a 900 secondi, la probabilità che si verifichi una fork di lunghezza 1 si attesta sul $6 * 10^{-4}$. La probabilità riscontrata nel modello è comunque molto maggiore rispetto a quella teorica dichiarata nell'articolo di presentezaione di Algorand [1] dell'ordine di 10^{-18}. Questa discrepanza può essere dovuta ad imprecisioni nel modello, principalmente nela gestione dei voti che in algorand riportano il mittente e il numero dello step `(r,p,s)` in cui sono stati prodotti e per questo vengono conteggiati di conseguenza, meccanica difficilmente riproducibile nell'attuale distribuzione di PRISM+.

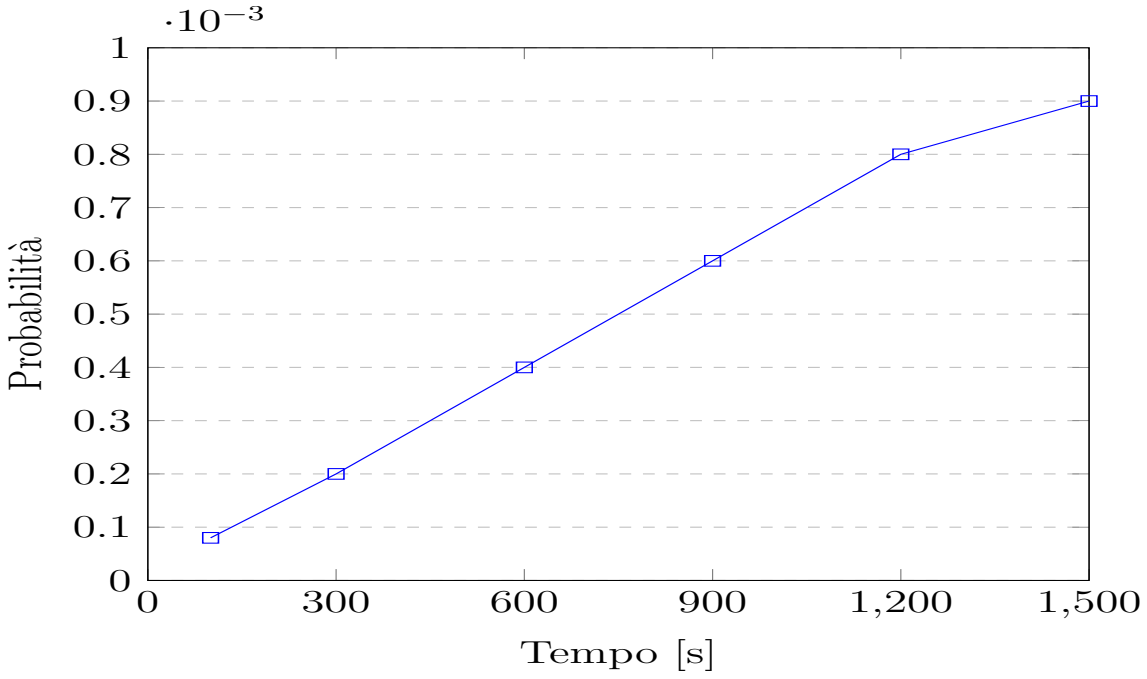

Figura 4.3: Probabilità in Algorand di una fork di lunghezza 1 in base al tempo

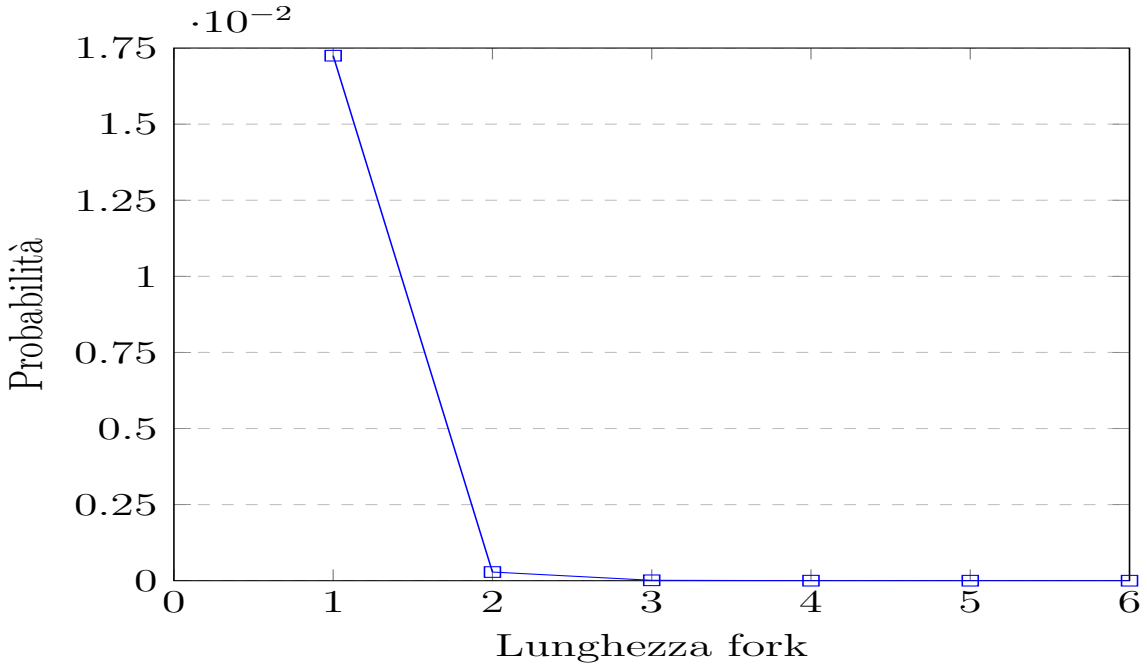

Figura 4.4: Probabilità in Bitcoin di una fork di lunghezza crescente

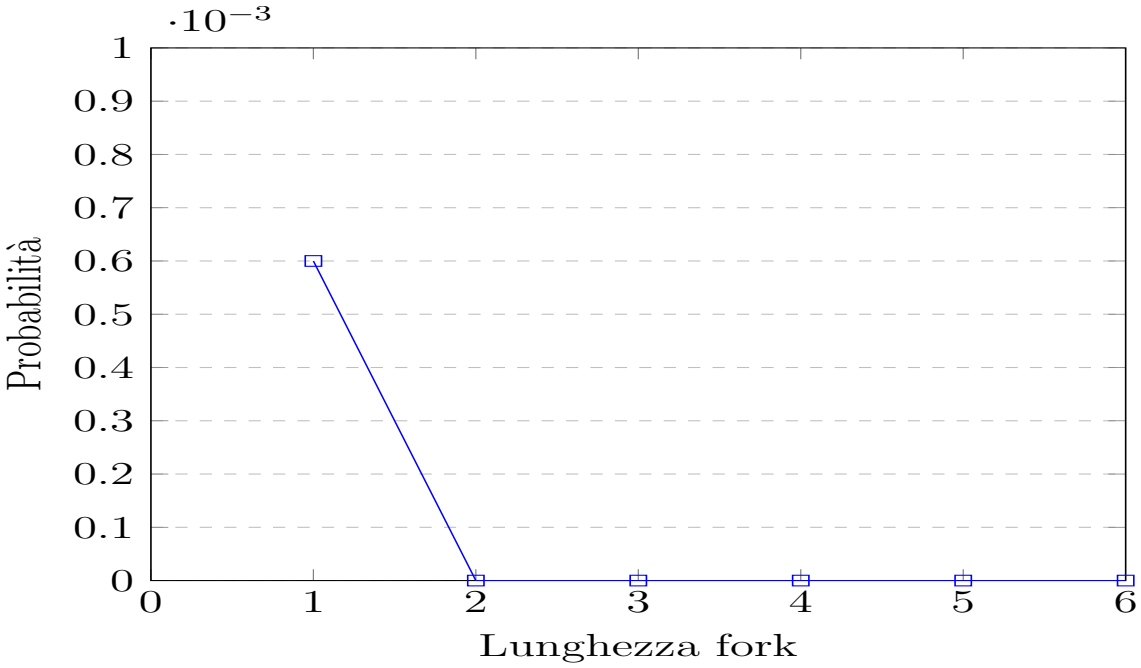

Figura 4.5: Probabilità in Algorand di una fork di lunghezza crescente

Capitolo 5

Modellizzazione di un possibile attacco : Partizione della rete

Dato che i messaggi all'interno della rete peer-to-peer di Algorand si propagano tramite *gossip* , ogni nodo trasmette i propri messaggi e quelli che riceve ai nodi a lui adiacenti, nel caso in cui la rete Intrnet subisse una partizione potrebbero verificarsi discrepanze tra i ledger delle due partizioni. In questo caso la partizione minoritaria non riuscirà a convalidare i blocchi e quindi un solo ramo della fork verrà esteso, per poi aggiornare i nodi rimasti indietro una volta risolta la partizione della rete grazie all'ultimo step descritto nel paragrafo su Algorand. Di seguito esamineremo il caso in cui gli 8 nodi del modello vengano divisi da un agente malevolo esterno in partizioni separate durante le fasi di voto.

5.1 Modifiche al modello

In questa versione del modello i nodi comunicheranno i propri voti ad un solo nodo della rete il codice per le votazioni modificato è descritto nel listing 5.1.

```
1  [softVote] (s = SoftVote) -> 1 : (softVotes0' = softVotes0 + stake)&(
     softVotes1' = softVotes1 + stake);
2  [softVote1] (s1 = SoftVote) -> 1 : (softVotes1' = softVotes1 + stake1
     )&(softVotes2' = softVotes2 + stake1);
3  [softVote2] (s2 = SoftVote) -> 1 : (softVotes2' = softVotes2 + stake2
     )&(softVotes3' = softVotes3 + stake2);
4  [softVote3] (s3 = SoftVote) -> 1 : (softVotes3' = softVotes3 + stake3
     )&(softVotes4' = softVotes4 + stake3);
5  [softVote4] (s4 = SoftVote) -> 1 : (softVotes4' = softVotes4 + stake4
     )&(softVotes5' = softVotes5 + stake4);
6  [softVote5] (s5 = SoftVote) -> 1 : (softVotes5' = softVotes5 + stake5
     )&(softVotes6' = softVotes6 + stake5);
```

```
7   [softVote6] (s6 = SoftVote) -> 1 : (softVotes6' = softVotes6 + stake6
     )&(softVotes7' = softVotes7 + stake6);
8   [softVote7] (s7 = SoftVote) -> 1 : (softVotes0' = softVotes0 + stake7
     )&(softVotes7' = softVotes7 + stake7);
```

Listing 5.1: Esempio di comandi di voto modificati per partizionare la rete

5.2 Analisi dati

In caso di partizioni della rete, si può notare che il tempo per approvare un nuovo blocco aumenta notevolmente (figura 5.1), infatti avremo la certezza di creare un blocco solo dopo 150 secondi e il tempo medio si attesta intorno ai 75 secondi, tempi comunque molto minori di quelli impiegati in Bitcoin. Il rallentamento nella approvazione dei blocchi è dovuto al numero limitato di voti che arrivano ad ogni nodo di conseguenza ognuno di essi dovrà eseguire più period all'interno dello stesso round per raggiungere il quorum di voti richiesto. Vediamo inoltre che la probabilità che si verifichi una fork di lunghezza 1 aumenta più lentamente in base al tempo(figura 5.2), cambiamento dovuto al minor numero di blocchi prodotto nello stesso tempo, e aumenta anche in valore quadagnando un ordine di grandezza come vediamo anche nella figura 5.3 in cui si nota un aumento aprezzabile anche nella probabilità che si verifichi una fork di lunghezza 2.

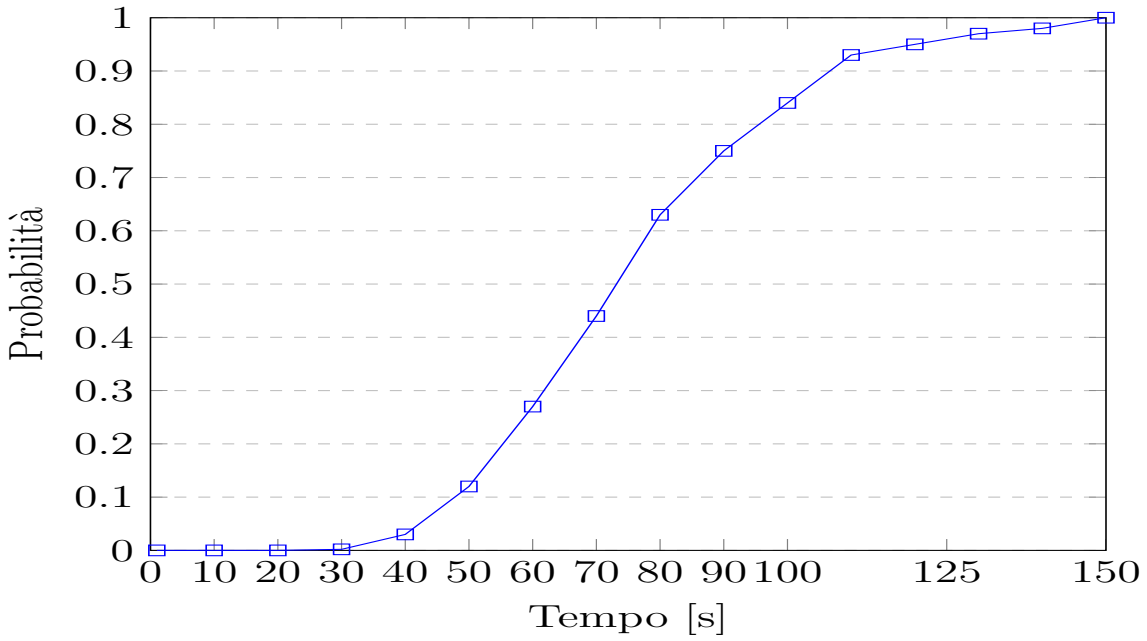

Figura 5.1: Probabilità che un nuovo blocco venga creato in tempo T con rete partizionata

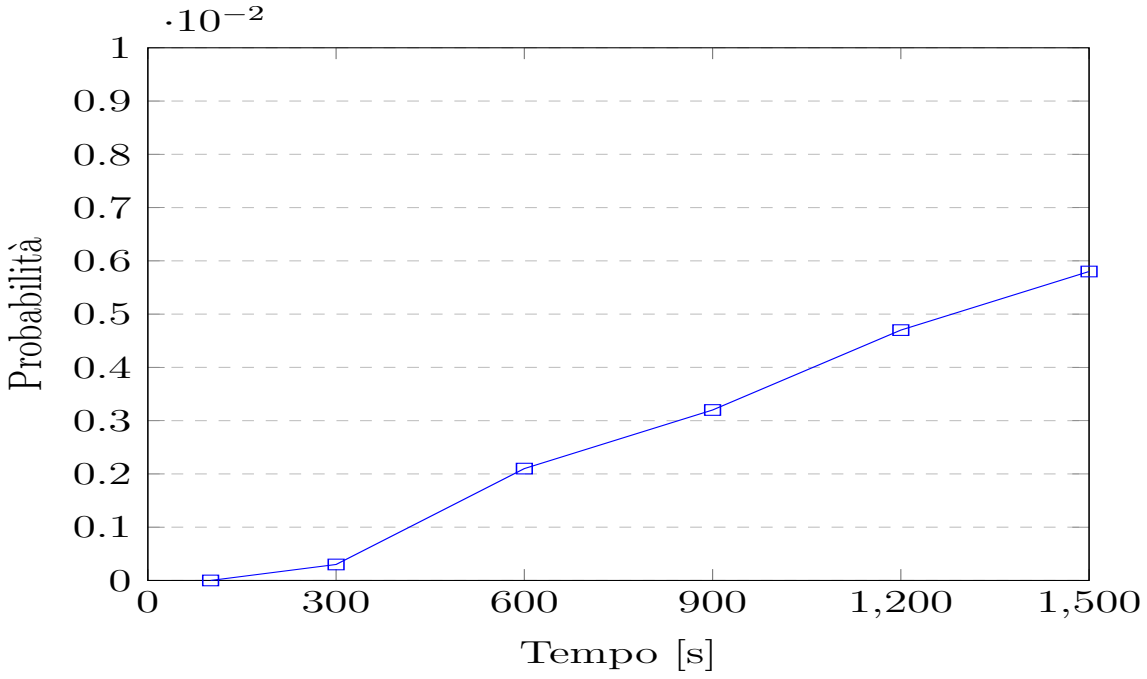

Figura 5.2: Probabilità di una fork di lunghezza 1 in base al tempo con rete partizionata

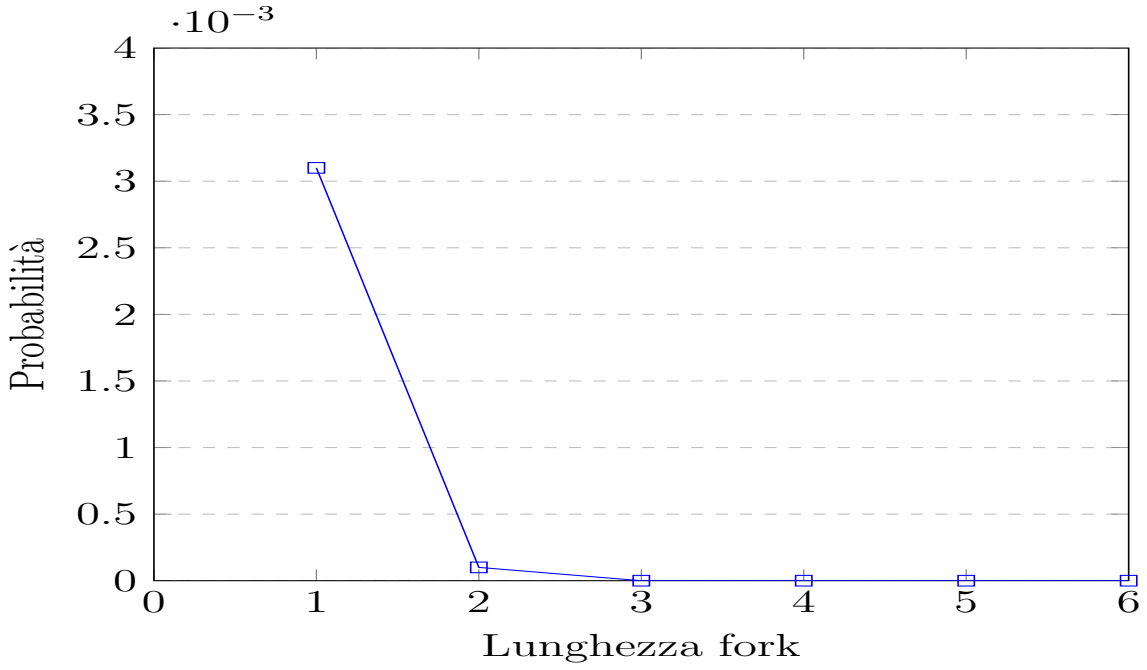

Figura 5.3: Probabilità di una fork di lunghezza crescentnte con rete partizionata

Capitolo 6

Conclusioni

La tecnologia blockchain sta prendendo sempre più piede come alternativa peer-to-peer distribuita ai metodi basati su di una attorità centrale. Questa tecnologia però, basandosi appunto su un sistema distribuito, ha bisogno di agenti esterni che partecipino alla rete. Per convincere gli utenti a partecipare spesso essi vengono premiati con l'emissione di moneta virtuale, le criptovalute, le transazioni eseguite con quosto tipo di moneta vengono salvate all'interno dei ledger della stessa blockchain. È dunque di fondamentale importanza valutare la robustezza ed efficienza di questi sistemi complessi preventivamente per proteggere le risorse in gioco.
In questa tesi abbiamo esaminato Algorand, una nuova blockchain che basandosi sull'approccio Proof of Stake prova a mitigare gli svantaggi dei precedenti approcci Proof of Work come la lentezza, la grande richiesta di risorse, energetiche e computazionali, e la difficile scalabilità. Modellizzando e testando l'algoritmo del consenso di Algorand, tramite il tool di analisi probabilistica PRISM e la sua estensione PRISM+, le dichiarazioni publicate nell'articolo di presentazione [1] sembrano reggere a meno di imperfezioni e semplificazioni del modello, garantendo una maggiore velocità, efficienza e scalabilità della struttura rispetto alle controparti Proof of Work. In futuro il modello potrà essere utilizzato come base per altri modelli di applicazioni basate su Proof of Stake su PRISM+, anche se per facilitare la creazione di modelli è auspicabile estendere ulteriormente PRISM+ con strutture per la gestione dei voti, di modo da poter salvare dei messaggi di voto contenenti informazioni come il blocco a cui si riferisce ed il mittente.

Listings

Elenco delle figure

Bibliografia

[1] J. Chen e S. Micali, «Algorand: A secure and efficient distributed ledger,» *Theoretical Computer Science*, vol. 777, pp. 155–183, 2019.

[2] M. Kwiatkowska, G. Norman e D. Parker, «PRISM 4.0: Verification of probabilistic real-time systems,» in *International conference on computer aided verification*, Springer, 2011, pp. 585–591.

[3] S. Nakamoto e A. Bitcoin, «A peer-to-peer electronic cash system,» *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, vol. 4, 2008.

[4] G. L. Cmandini. (). «Silvio Micali, il genio dell'informatica che vuole rendere più efficiente la blockchain,» indirizzo: `https://forbes.it/2018/10/16/silvio-micali-blockchain-algorand/#!`.

[5] Y. Gilad, R. Hemo, S. Micali, G. Vlachos e N. Zeldovich, «Algorand: Scaling byzantine agreements for cryptocurrencies,» in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[6] A. foundtion. (2019). «Algorand blockchain features,» indirizzo: `https://github.com/algorandfoundation/%20specs/blob/master/overview/Algorand%20v1%20spec-2.pdf`.

[7] S. Mehlhorn. (2019). «Demystifying Algorand Rewards Distribution: A Look at How and When Algorand Token Rewards Are Calculated,» indirizzo: `https://www.purestake.com/blog/algorand-rewards-distribution-explained/`.

[8] M. Kwiatkowska, G. Norman e D. Parker, «Stochastic model checking,» in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, Springer, 2007, pp. 220–270.

[9] S. Bistarelli, R. De Nicola, L. Galletta, C. Laneve, I. Mercanti e A. Veschetti, «Stochastic Modelling and Analysis of the Bitcoin Protocol,» Manuscript under review, 2021.

[10] Y. Liu, «Investigating Byzantine Agreement Consensus Algorithm of Algorand,» tesi di dott., 2020.

[11] M. A. Alturki, J. Chen, V. Luchangco, B. Moore, K. Palmskog, L. Peña e G. Roşu, ≪Towards a verified model of the algorand consensus protocol in coq,≫ in *International Symposium on Formal Methods*, Springer, 2019, pp. 362–367.