Vol.2020-DPS-185 No.11
2020/12/22

Ethereumに基づいたアプリケーションの実行時間定式化の検討と計測

阿部 涼介[1,a)]　鈴木 茂哉[1]

概要：Ethereum は，P2P ネットワーク上で改ざん困難な台帳を信頼する第三者なく形成するシステムであるブロックチェーン上でプログラムを実行できるプラットフォームである．Ethereum 上に構築されたアプリケーションの操作に要する時間は，そのアプリケーションが実用に耐えうるか判断するための重要な要素である．先行研究では，限られたネットワーク参加者を想定したブロックチェーン自体の性能を検証が行われているが，Ethereum のような開かれたネットワーク上での伝播遅延を考慮した検証，およびアプリケーションの実行時間の定式化には至っていない．本研究では，Ethereum の動作を精査し，その上に構築されるアプリケーションの実行に要する時間の定式化を試みる．定式化された実行時間を検証するために，遅延をエミュレーションしたプライベートなネットワーク，試験環境であるテストネットワーク上で実験を行った．実験の結果，一定以上の伝播遅延が発生する時に遅延以上のオーバヘッドが存在することが示唆された．本研究の成果を基にさらなる高精度な定式化を行うことで，Ethereum 上に構築されるアプリケーションの実行時間を想定し，実用に耐えうるか事前に検証可能となることが期待される．

Formalize and Measurement Duration for Operating an Application Based on Ethereum

1. はじめに

Ethereum はブロックチェーン上にチューリング完全なプログラムを記述し，実行することのできるアプリケーションプラットフォームである [1]．ブロックチェーンは P2P ネットワーク上で改ざん困難な台帳を信頼する第三者なく形成するシステムであり，暗号通貨 Bitcoin の基幹技術として発明された [2]．その特性は，暗号通貨のみならず様々な分野へ適用が期待され，研究が盛んに行われている．

ブロックチェーン上に構築されたアプリケーションが実用に耐えうるか検討する一つの指標として，アプリケーションの操作が完了するまでの時間（実行時間）が考えられる．尾根田らは，Ethereum だけでなく Hyperledger fabric，Quoram のブロックチェーン基盤ソフトウェアとしての性能評価を試みた [3]．しかし，尾根田らの検証においては，ネットワーク参加者が限られた中で構成するブロックチェーンのみを対象にしているが，Ethereum は本来ネットワークへの参加を制限しない設計である．不特定多数の参加者が存在し，全世界で 1 つのオーバーレイネットワークを構築するため，データ伝播の遅延が発生する．こうした遅延を前提としたブロックチェーンへの攻撃や，シミュレーションなどの研究が盛んに行われている [4], [5]．

本研究は，不特定多数の参加者の存在するブロックチェーン上に構築されたアプリケーションの実行時間の定式化と計測を行い，そうしたアプリケーションが実用に耐えうるか検討するための指標を提供することを目的とする．Ethereum を対象にその動作の精査を行い，定式化を試みた．また，定式化された実行時間を検証するために，遅延をエミュレーションしたプライベートなネットワーク，Ethereum の試験環境であるテストネットワークの一つである Ropsten ネットワーク上で実験を行った．

本研究を基に高精度な実行時間の推定を行うことで，ブロックチェーン上に構築されるアプリケーションの実行時間を推定することが可能となり，それらを考慮した上で精密な技術選定の検討が可能になると期待される．

[1] 慶應義塾大学大学院 政策・メディア研究科
Keio University Graduate School of Media and Governance, Fujisawa, Kanagawa 252–0882, Japan
[a)] chike@sfc.wide.ad.jp

2. Ethereum

2.1 Ethereum 概要

Ethereum では，P2P ネットワーク上でシステム上にハードコードされた仮想通貨 “Ether” の送金と，“スマートコントラクト” と呼ばれるプログラムの実行を扱う．各操作は “トランザクション” と呼ばれるデータで表現される．トランザクションは，その発行者の保有する秘密鍵によるデジタル署名と，当該トランザクションにおける操作の情報が含まれる．トランザクションは “ブロック” と呼ばれる単位で各ノードに “ブロックチェーン” と呼ばれる台帳内に保存される．ブロックにはトランザクションのリストとブロックチェーン上で直前のブロックのハッシュ値が含まれる．

ユーザがトランザクションを作成すると，そのトランザクションを Ethereum ネットワークに参加するノードへ送信する．受信したノードは，当該トランザクションの操作の正当性とデジタル署名の検証を行う．検証に成功すると，ノードは一度トランザクションを “TXPool” と呼ばれるメモリ上の領域に保存し，他の Ethereum ネットワーク上のノードにブロードキャストする．トランザクションを受信したノードは，同様に検証と TXPool への保存を行う．ネットワーク上の任意のノードは自身の持つ TXPool に存在するトランザクションからリストを作成し，ブロックチェーンの最新ブロックのハッシュ値を算出することでブロックを作成する．この時，ブロックを作成するノードは “Proof-of-Work” と呼ばれるプロトコルを用いて，一定の計算パワーを投入し計算を行ったことの証明を行う [6]．ブロックを作成すると，当該ノードはブロックをブロードキャストし，受信したノードはブロックに含まれるトランザクションと直前のブロックのハッシュ値，ブロックが計算パワーが投入されて作成されたことを検証する．ブロックの検証に成功すると，受信したノードはブロックをブロックチェーンに追記する．ブロックを作成する際に計算コストを投じるインセンティブとして，ブロックを作成したノードは一定量の Ether を受け取ることができる．このインセンティブ構造を金の採掘に擬え，ブロックの生成を “マイニング” と呼び，マイニングを行うノードを “マイナー” と呼ぶ．

同じブロックを参照するブロックが同時に複数のマイナーによって作成されるなど，矛盾する複数のブロックが存在する場合がある．この状態を “Fork” と呼ぶ．Fork を解決するために，ノードは当該ブロックの生成において最も大きな計算パワーが投入されたブロックを正規のブロックとして採用する．この仕組みにより，もし攻撃者が過去のブロック中のデータの改ざんを試みると，当該ブロックに連なるブロック中のハッシュ値が変化し，連鎖的に最新のブロックまで変化することになる．従って，攻撃者は改ざんするブロックから最新のブロックまで全てのブロックを計算パワーを投入して再作成しなければならない．しかし，攻撃者が改ざんのためのブロックの生成を行なっている最中にも，他のノードは改ざん前のブロックチェーンに追記するようにブロックを生成している．そのため，攻撃者は改ざん前のブロックチェーンを追い越すために，ネットワーク全体に投入されている 50%以上の計算パワーを持たない限り改ざんを必ず成功させることはできない．ネットワーク上に十分な計算パワーが投入されていればこの攻撃を実施することは困難であるため，ブロックチェーンは改ざん困難であると考えられる．ここまで述べた動作はブロックチェーン全体を保持することで，他のノードに依存することなく実行可能であるため，信頼する第三者を置かずに動作させることが可能である．

2.2 コントラクトの操作

図 1 に Ethereum のコントラクト操作の概要を図示する．コントラクトに関わるトランザクションには，コントラクトの操作に関わるデータが含まれる．当該トランザクションがブロックチェーン に格納されると，各ノード上に存在する Ethereum Virtural Machine（EVM）上でコントラクトが実行される．コントラクトをデプロイする際は，専用のプログラミング言語で記述されたコントラクトのソースコードをコンパイルしたデータを含むトランザクションを発行する．当該トランザクションがブロックチェーンに格納されると，“コントラクトアドレス” と呼ばれる識別子が発行され，コントラクトが操作可能な状態となる．コントラクトがデプロイされると，コントラクトに関わる変数の値を格納するための Contract State が EVM 上に作られる．Contract State を更新するようなコントラクトの操作を行う際は，コントラクトアドレスとコントラクトの関数に対する引数をトランザクションに含めたトランザクションを発行する．当該トランザクションがブロックチェーンに格納されると，EVM は Contract State を読み出し，処理を実行する．Contract State を読み取るだけの操作を行う際はトランザクションを発行せずとも実行することができる．

2.3 コントラクトの実行手数料と例外処理

Contract State を更新するトランザクションは，実行のための手数料を Ether で支払う必要がある．この手数料は “Gas” と呼ばれる単位で表現される．Gas の量は，当該トランザクションの操作により EVM 上で実行される OPCODE の実行ステップ数によって決定される．マイナーはコントラクト実行のトランザクションを含むブロックを作成した時，含まれるトランザクションの実行手数料を通常のマイニング報酬に加えて得ることができる．

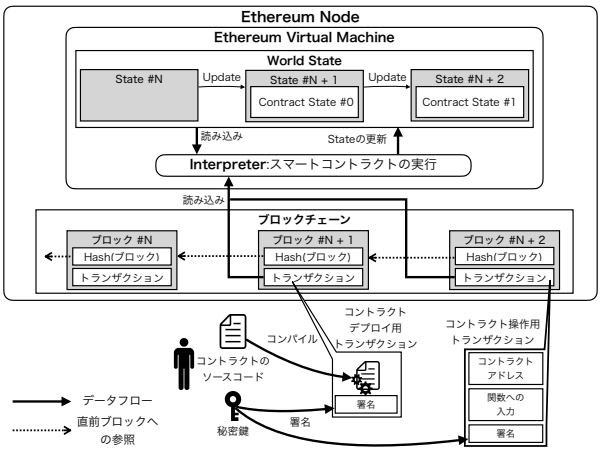

図 1 コントラクト操作概要
Fig. 1 Overview of Ethereum Contract Operation

コントラクト実行の際は，1Gas あたりの Ether の価格である “GasPrice” と当該トランザクションで消費する Gas 数を指定する．当該トランザクションが指定した Gas を全て消費しなかった場合，Gas の残額は実行者に返金され，実行分のみがマイナーの報酬となる．ただし，コントラクトの実装の中で例外処理が記述され，実行時に例外処理が発生した場合，トランザクションは正常にブロックチェーンに格納された上で Gas は全てマイナーに支払われる．

3. 想定するアプリケーション

Ethereum ベースのアプリケーションの実行時間を計測するにあたり，本研究では任意のトークンを発行するための標準規格である ERC-20 を対象として実験を実施する [7]．ERC-20 は特定のアプリケーションではあるものの，そのために必要な操作シーケンスは Ethereum ベースのアプリケーション一般に適用可能なものであると考えられる．

3.1 ERC-20

ERC-20 は，Ethereum 上で任意のトークンを発行するための標準規格であり，主として **表 1** に示す関数が定義されている．

ERC-20 で特徴的な部分は，「他者へ送金の実行委託を行う」という関数が定義されている点である．**図 2** に送金委託のプロセスを示す．Carol は ERC-20 コントラクト上で定義されたトークンを保有しており，その一部を Alice に送金の委託を行う．まず最初に Carol は Alice に対して approve 関数で委託する金額を指定する．approve 関数が実行されると，Alice を含む任意のユーザは allowance 関数を実行することで Carol が Alice に送金委託しているトークン量を確認することができる．Alice は十分なトークン量が送金委託されたことを確認した後，transferFrom 関数を実行することで，Carol の持つトークンを Bob に対して送金を行うことができる．trasferFrom 関数実行時に，Alice が送金しようとした金額が Carol によって送金委託されていなかった場合，例外処理が実行され，送金は実施されない．

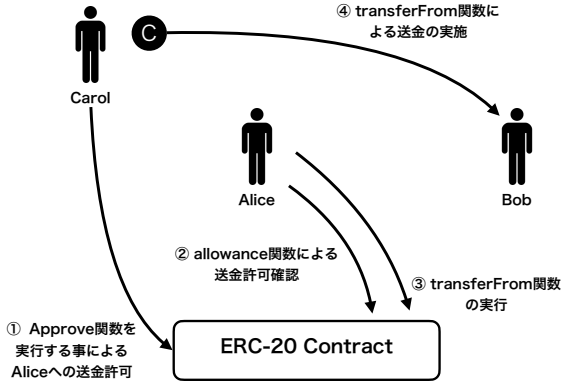

図 2 ERC-20 における送金委託と送金のプロセス
Fig. 2 A Process of TransferFrom on ERC-20

本研究では，アプリケーションの動作の例として，ERC-20 における transferFrom 関数を使った送金を取り上げる．この操作において，送金実行者 (Alice) は，送金を行いたい金額を送金委託元 (Carol) が送金委託しているかを確認した上で transferFrom を実行しなければ，十分な量委託されてないことで例外処理が行われ，トランザクション手数料を消費してしまう可能性がある．そのため，送金実行者はこの操作を行う際，Contract State を監視し，実行可能な状態であることを確認した上で transferFrom 関数を実行する必要がある．このコントラクトの状態を確認し特定の状態であればコントラクトの関数を実行するというモデルは ERC-20 に限らず多くの Etehreum 上に構築されるアプリケーションに共通した操作であると考えられる．

3.2 アプリケーションのエンティティ

Ethereum ベースのアプリケーションでは，以下のようなエンティティが存在すると考えられる．

- **Gateway ノード**: Ethereum ネットワークに参加し，ブロックなどの検証を行う Ethereum ノード
- **コントラクト**: 上記ノードが参加するネットワークのブロックチェーン上にデプロイされているコントラクト
- **クライアント**: 上記コントラクトの状態に応じてコントラクトの関数を操作するトランザクションを投入するクライアント

Ethereum では，ブロックチェーン全体を保持するノードは他のノードに依存することなく，独立して動作可能である．しかし，アプリケーションのエンティティであるク

表 1　ERC-20 コントラクトにおける主要な関数
Table 1　Functions in ERC-20 Contract

関数名	引数	概要
transfer	送金先，金額	送金先へ指定金額のトークンを送金する
approve	送金委託先，金額	送金委託先へ指定金額まで実行者のトークンの transferFrom による送金を許可する
transferFrom	送金元，送金先，金額	事前に approve によって実行者が送金元から許可を得た上で送金元から送金先へ指定金額のトークンを送金する
allowance	トークン保有者，送金実行者	トークン保有者の持つトークンのうち，どれだけのトークン量が送金実行者に approve されているかを確認する

ライアントはユーザが直接操作を行うデバイスであり，スマートフォンなどのストレージ容量が限られたデバイスであると考えられる．2020 年 11 月現在，Ethereum メインネットワークのブロックチェーンのデータサイズは 550GB を超えており，今後も単調に増加することを考慮すると，ストレージ容量の限られたデバイスで保持し続けることは困難である．そのため，クライアントは特定の Ethereum ノードを Ethereum ネットワークへの Gateway として利用し，ブロックチェーン上に書き込まれた情報などは，その Gateway ノードから取得するアーキテクチャを取らざるを得ない．このとき，当該 Gateway ノードがクライアントからの問い合わせに対してブロックチェーン上とは異なる情報を応答したとしても，クライアントは偽りであることを検知することは難しい．そのため，クライアントは Gateway ノードを信頼する必要性がある．

3.3　アプリケーションの動作

本節では，アプリケーションの動作を概観する．**図 3** に，操作時の動作シーケンスを示す．本研究で対象とする ERC-20 コントラクトにおける transferFrom 関数での送金操作を行うためには，送金実施者のクライアントは十分な金額の送金委託を受けているかどうかを確認する必要がある．そのため，クライアントは Contract State のポーリングを行う．次に，十分な金額が送金委託されたことを確認すると，クライアントは transferFrom 関数を実行するためのトランザクションを作成し，Gateway ノードを通じて Ethereum ネットワークへと投入する．その後，クライアントは当該送金が完了したことを確認するために再度 Contract State をポーリングし，実行の完了を待つ．Gateway ノードはトランザクションを受け取ると，検証を行った後，トランザクションを Ethereum ネットワーク上の他のノードへブロードキャストする．Ethereum ネットワーク上の任意のマイナーが当該トランザクションを含んだブロックの作成に成功すると，マイナーは当該ブロックをブロードキャストする．クライアントが参照している Gateway ノードが当該ブロックを受信すると，ブロックの検証を行い，ブロックチェーンへの追記を行う．ブロックチェーンへの追記が行われた後，Contract State が更新され，クライアントは transferFrom 関数が正常に実行されたことを確認できる．本研究では，このプロセスのうち，クライアントによるトランザクションの投入から実行が確認されるまでを実行時間として扱う．

4.　実行時間の定式化

本節では 3 節で述べたアプリケーションの実行時間の構成要素と，その影響について述べる．

4.1　定式化

クライアントからトランザクションによって状態が更新されたことが観測可能となるためには，トランザクションがブロックに含まれる必要がある．トランザクションがブロックに含まれるまでの時間 ($Duration_{tx}$) は、以下のようなパラメータに影響される．

- $Duration_{block}$: ブロックの生成間隔
- $BlockNum_{untilInluded}$: 投入したトランザクションがブロックに格納されるまでのブロック数
  - N: トランザクション投入時の Gateway ノード上最新ブロック高
  - M: トランザクションを含むブロック高
  - $BlockNum_{untilInluded} = M - N$

また，トランザクション投入時には，Gateway ノード上の最新ブロックがマイニングされてからの経過時間が存在することを考慮すると $Duration_{tx}$ は以下の式で算出される．

$$Duration_{tx} \leq Duration_{block} \cdot BlockNum_{untilInluded}$$

4.2　パラメータの決定要素

本項では，本節で述べた実行時間が影響を受ける要素を検討する．

4.2.1　$Duration_{block}$

ネットワーク依存のパラメータであり，Ethereum メインネットワークにおいては平均 12 秒になるようにアルゴリズムによって調整されている．

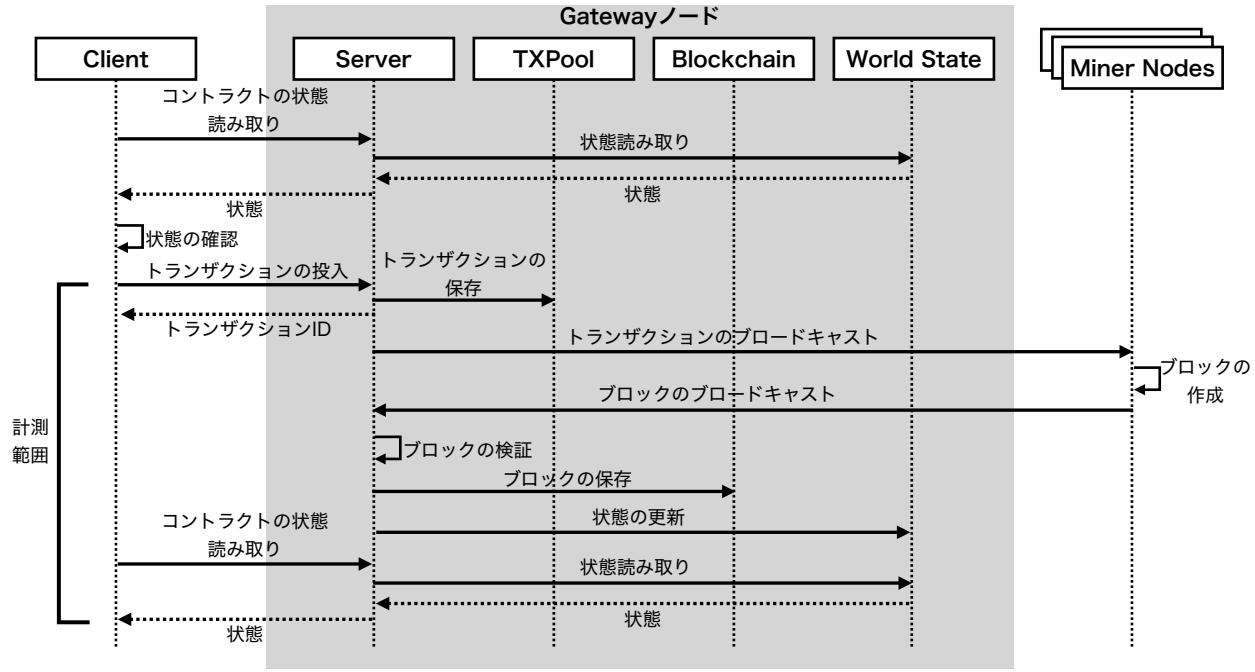

図 3 アプリケーション操作のシーケンス
Fig. 3 A Sequence of Operation of the Application

4.3 $BlockNum_{untilInluded}$

マイナーが当該トランザクションをブロックにどのタイミングで含めるかに依存し，支配的な要素は以下の 2 つである．

- トランザクションが投入された後，ブロックを作成するマイナーへどれだけ早くトランザクションが伝播するか
- トランザクションがマイナーにとってブロックに含めるに足りる十分な手数料が支払われるか

Ethereum メインネットワークにおいては，マイニング事業者が大きな計算パワーを投入して参入しており，これらに対抗して Gateway ノードでマイニングを行なっても，マイニング事業者よりも早くブロックの生成に成功する可能性は低い．そのため，トランザクションを投入した時，マイニング事業者のノードまで当該トランザクションがいち早く伝播することで，トランザクションが早期にブロックに含まれる可能性が高くなる．

一方，マイナーはブロックを作成した時，マイニング報酬とブロック内のトランザクション手数料の合計値を得ることができる．従って，一般的にマイナーは TXPool 上のトランザクションからトランザクション手数料が最大化されるようにトランザクションを選択してブロックを作成する．そのため，トランザクション生成者は適当な金額となるように GasPrice を調整し，適当な時間でブロックに格納されるように試みる．Ethereum の主要な実装の 1 つである Geth の実装の中には，適当な金額の推定のために Gas Price Oracle（GPO）と呼ばれるアルゴリズムが実装されている．近年の調査によれば，GPO で GasPrice を設定したトランザクションの $BlockNum_{untilInluded}$ は平均 2 となっている [8]．

5. 実験

本節では， 4 節で述べた定式化を試みた実行時間を検証するために行った実験について述べる．実験のために，ERC-20 コントラクトの操作を行うクライアントを Python で CLI アプリケーションとして実装した．その上で，伝播遅延をエミュレーション可能な環境としてプライベートネットワークを Docker コンテナを用いて構築し，実験を行った．その後，Ethereum の試験環境の一つである Ropsten ネットワークを用いて，実際に遅延が存在する全世界に広がるネットワーク上での実験を行った．本実験に用いたソフトウェアおよびハードウェアを **表 2** に示す．同一のハイパーバイザ上で動作させることによるオーバヘッドを低減するために，ハイパーバイザには十分な CPU およびメモリを搭載した．

5.1 プライベートネットワークでの実験

ネットワークの遅延をエミュレートした環境での実験として，ローカルにプライベートなネットワークの環境を構築し，実験を行った．0ms から 800ms まで 200ms 毎に遅

表 2　実験環境環境
Table 2　Configuration of Containers

構成要素	実装
コンテナイメージ	ethereum/client-go:v1.9.24
コンテナハイパーバイザ	Docker version 19.03.13
OS	Debian 10
仮想ハードウェア	22 vCPUs, 90 GB Memory
ハイパーバイザ	ESXi 6.5
ハードウェア	Fujitsu rx200 s6 (96GB Memory, 24CPUs)

延の発生するプライベートネットワークをそれぞれ構築し，図 3 に示したシーケンスを 100 回計測した．

5.1.1　実験環境

実験のために，**図 4** のようなネットワークを構築した．各 Ethereum ノードは docker コンテナで Gateway ノード 1 コンテナ，マイナーノード 3 コンテナを構築した．加えて，Gateway ノードとマイニング事業者間の遅延をエミュレーションするために，Gateway ノードと各マイニングノードの間に “Boot ノード” を配置しスター型のトポロジをした上で，Boot ノード上で tc コマンドによって遅延をエミュレーションした．Boot ノードはマイニングには参加せず，Gateway ノードから各マイナーへ伝播を中継する役割を担う．本環境では，$Duration_{Block}$ は平均 2 秒程度であった．

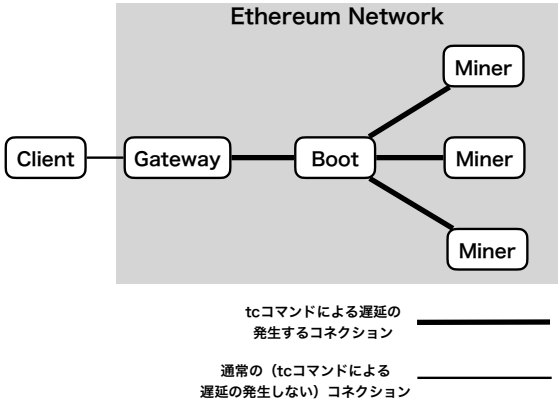

図 4　遅延をエミュレーションしたプライベートネットワーク
Fig. 4　A Private Network with Delay

5.1.2　実験結果

図 5 に実験結果を示す．遅延の大きさに応じて実行時間が長くなることが示されている．次に，遅延が存在しない際の結果の中央値をベンチマークとして各遅延発生時の時間を推定し，比較を行った．図 4 に示したネットワーク上で，Gateway ノードから各マイナーへ Boot ノードで発生させた遅延が発生する．図 3 で示したようなトランザクションが伝播した後ブロックが伝播するシーケンスの中では，Gateway ノードと Boot ノード間，Boot ノード間と Gateway ノード間でそれぞれトランザクションとブロックの伝播において遅延の影響を受ける．そのため，遅延が存在しない場合の中央値をベンチマークとし，ベンチマークに遅延値の 4 倍の値を加えたものをそれぞれ遅延が存在するネットワーク上での推定値とした．

推定値と結果を比較すると，遅延が大きくなるほど遅延分以上に実行時間が大きくなる傾向が存在する．これは，各マイナーに到達するまでに $Duration_{Block}$ 以上の遅延が発生したため，トランザクション投入直後のブロック作成までにマイナーに到達しなかったためであると考えられる．各遅延発生時の $BlockNum_{untilIncluded}$ の中央値は，遅延が 400ms 以下の時 2 ブロックであったが，600ms 以上の際には 3 ブロックとなった．

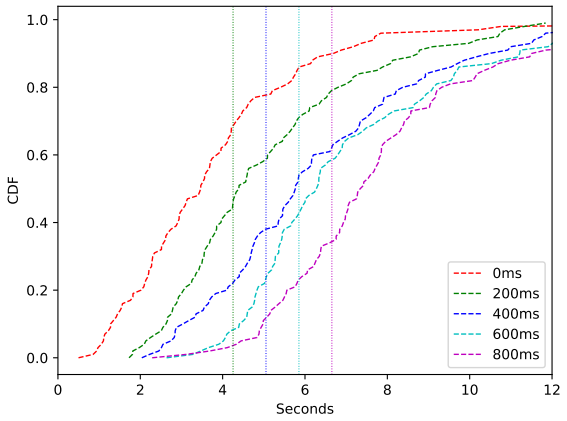

図 5　プライベートネットワークでの結果；各垂直線は遅延がない場合の結果の中央値と各遅延から計算した推定値
Fig. 5　Results on the Private Network; Vertical lines are estimated values that are estimated by a result with no delay and each delay.

5.2　Ropsten ネットワークでの実験

次に，実際に全世界で展開されている Ethereum の試験環境である Ropsten ネットワークで同様に 100 回の計測を行った．Ropsten ネットワークは，メインネットワークと同様に平均 12 秒に 1 回ブロックが生成されるように調整されている．メインネットワークとの違いは，Ropsten ネットワーク上で流通している仮想通貨には価値がつかないとされている点であり，その他の動作はメインネットワークと同一である．

5.2.1　実験環境

実験を行うために、ローカル環境下に Docker コンテナで Ropsten ネットワークに参加した Gateway ノードを構築した．Ropsten ネットワークでは全世界にマイナーが存在し，実際のインターネット上での遅延が発生するため，特段の遅延は加えずに実験を行った．実験環境のネット

ワーク概要を 図 6 に示す．

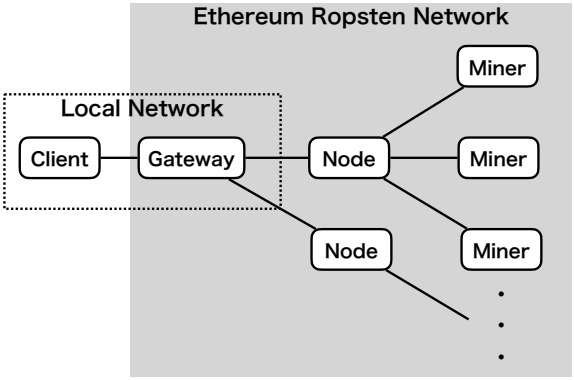

図 6　Ropsten ネットワークでの実験環境
Fig. 6　An Expetimental Enviroment on Ropsten Network

5.2.2　実験結果

Ropsten ネットワークにおける実験結果を 図 7 に示す．Ropsten ネットワークにおいて，マイナーがブロックを作成した正確な時刻を計測することは困難であるため，Gateway ノードがブロックを受信した時間をブロック生成時刻として $Duration_{Block}$ を算出した．本実験中の $BlockNum_{untilIncluded}$ の中央値は 1 であり，実験期間中の $Duration_{Block}$ は 18.04 秒であったことから推定値を算出した．

実験結果と推定値を比較すると，推定値よりも実行時間がやや長い傾向が存在する．この推定値は遅延を考慮していないため，全世界に展開される Ropsten ネットワーク上では，ある程度遅延の影響を受けることが理由として考えられる．

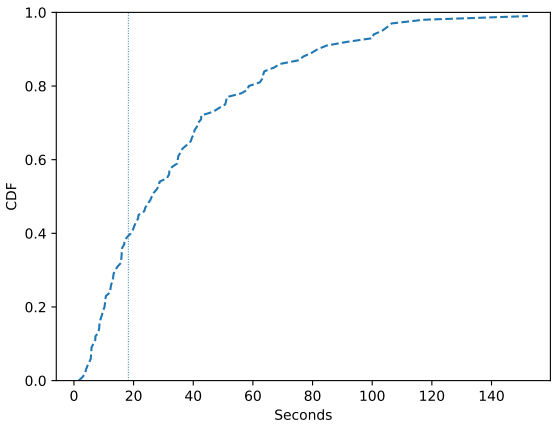

図 7　Ropsten ネットワークでの結果；垂直線はブロック生成間隔から計算した推定値
Fig. 7　Results on the Private Network; Vertical lines are estimated values that are estimated by a duration of block creation

5.3　実験のまとめ

本実験では，遅延を加えたプライベートネットワークと，Ropsten ネットワークでの実験を行った．プライベートネットワークでの実験では，一定以上の遅延を加えると遅延分以上のオーバヘッドが発生することが示唆された．4 節で述べた定式化に遅延への考慮を加えることで，より正確な実行時間の推定が可能になることを示唆している．Ropsten ネットワークでの実験では，遅延を考慮していない推定値よりも実行時間がかかることから，遅延の影響が実際に全世界に広がるネットワークでは存在することを示唆している．

6.　今後の展望

本節では，本検討の今後の展望を述べる．

6.1　Ethereum の Sharding 適用への対応

Ethereum は未だ不完全な技術であり，本研究で取り上げたアプリケーションの実行時間においても，一般的なクライアントサーバモデルのアプリケーションに比較して，非常に性能が低い．そのため，Ethereum 開発コミュニティは，ブロックチェーンを並列して動作させることでスケーラビリティを確保する “Sharding” と呼ばれる手法を導入するべく，開発を進めている．Sharding が導入されれば Ethereum の動作は大きく変更されるため，本研究で示した定式化の検討だけでは今後の Ethereum 上でのアプリケーションの実行時間を検証することは難しい．Sharding を前提とした Ethereum のシミュレータの開発などが進んでおり，それらと組み合わせながら検討を進めていく必要がある [9]．

6.2　Fork 発生の勘案

本研究での実験では，トランザクションがブロックに取り込まれるまでを実行時間として計測したが，Fork の発生を勘案していない．ブロックチェーンで用いられる Fork の解決手法は，あるノードが一度正規のブロックとして認めた後により計算パワーが多く投入されているブロックが発生し，正規のブロックではなくなる可能性が常に存在する．Proof-of-Work を用いたブロックチェーンでは，あるトランザクションを含むブロックの上に数ブロック積み上がることで正規のブロックでなくなる可能性が低くなる．そのため，数ブロック積み上がった段階で実行が “確定されたという見なし” をする運用が一般的である．Ethereum では，その見なしを行ったブロックを “Canonical ブロック” と呼ぶ．より実運用における実行時間を反映した定式化と計測を行うためには，Canonical ブロックに至るまでのプロセスを勘案する必要があるだろう．

また，伝播遅延によって Fork が発生する可能性が影響を受けることが先行研究から明らかになっている [10]．こ

れらの取り組みや，ブロックチェーンネットワークにおける伝播遅延推定の試みを組み合わせることで，より精度の高い実行時間の推定が行えると考えられる [11].

6.3 Ethereum アプリケーション開発プロセスの定式化

本研究では，遅延が Ethereum 上に構築されるアプリケーションの実行時間に影響を与えることが示唆された．アプリケーションを開発する際には実際の本番環境であるメインネットワークで動作させることを想定してテスト・開発を行なっていく必要がある．Ropsten ネットワークは同様の目的のために利用可能であるが，全世界に開かれたネットワークであるため，並列して様々な実験が行われており，その影響がどのように存在するかは明らかとなっていない．安定して開発を進めるためには，試験環境として閉じていながらも，遅延などをエミュレーションしメインネットワークをある程度想定した試験ができることが好ましい．従って，そうした試験環境のオーケストレーションツールなどの開発が今後の課題として挙げられるだろう．また，それらを利用しながら，開発プロセスの定式化を行うことで，安定して実運用可能なアプリケーションを構築することが可能になると考えられる．

7. おわりに

本研究では，Ethereum 上に構築されるアプリケーションの実行時間の定式化を試みた上で，プライベートネットワークと Ropsten ネットワークにおいて実験を行った．プライベートネットワークでの実験では，一定以上の伝播遅延が発生することで伝播遅延分以上のオーバヘッドが存在することが示唆された．また，Ropsten ネットワークでの実験では，遅延が実際に全世界に展開されるネットワークでは影響を生んでいる可能性が示唆された．本研究による実行時間の定式化が進むことで，Ethereum 上で構築されるアプリケーションが実行時間の視点から実運用に耐えうるかを検証することが可能となることが期待される．より精度の高い定式化を行うためには，ネットワーク全体での伝播遅延推定を取り入れるなどの検討を進めることなどが今後の課題である．

参考文献

[1] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014).
[2] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2019).
[3] 尾根田倫太郎，秋田佳記， 何岩彬， 竹林陽，小倉拓人，鈴木貴之ほか：ブロックチェーン基盤ソフトウェア性能検証— Hyperledger Fabric, Quorum, Ethereum の横串比較—，デジタルプラクティス，Vol. 10, No. 3, pp. 506–523 (2019).
[4] Apostolaki, M., Zohar, A. and Vanbever, L.: Hijacking Bitcoin: Routing Attacks on Cryptocurrencies, *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 375–392 (2017).
[5] Aoki, Y., Otsuki, K., Kaneko, T., Banno, R. and Shudo, K.: SimBlock: A Blockchain Network Simulator, *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 325–329 (2019).
[6] Jakobsson, M. and Juels, A.: Proofs of Work and Bread Pudding Protocols(Extended Abstract), *In Proc. Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99)*, pp. 258–272 (1999).
[7] Vogelsteller, F. and Buterin, V.: ERC-20 token standar (2018).
[8] Spain, M., Foley, S. and Gramoli, V.: The Impact of Ethereum Throughput and Fees on Transaction Latency During ICOs, *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*, Vol. 71, pp. 9:1–9:15 (2020).
[9] : Shargri-La A Sharded Blockchain Simulator (2020).
[10] Otsuki, K., Aoki, Y., Banno, R. and Shudo, K.: Effects of a Simple Relay Network on the Bitcoin Network, *Proceedings of the Asian Internet Engineering Conference*, AINTEC '19, New York, NY, USA, Association for Computing Machinery, p. 41–46 (2019).
[11] Kanda, R. and Shudo, K.: Estimation of Data Propagation Time on the Bitcoin Network, *Proceedings of the Asian Internet Engineering Conference*, AINTEC '19, New York, NY, USA, Association for Computing Machinery, p. 47–52 (2019).