令和四年 三月

卒業論文

スマートコントラクトを活用した分散型電子書籍流通システムの設計と試作

広島大学
情報科学部情報科学科
分散システム学研究室

尾形　啓悟

指導教員： 藤田　聡 教授

卒業論文概要 2022 年度（令和四年度）

スマートコントラクトを活用した
分散型電子書籍流通システムの設計と試作

今日の電子書籍サービスはその中央集権的なシステムに起因してサービス終了やユーザアカウント停止などにより電子書籍が読めなくなる、サーバに負荷が集中してサービス全体が停止するなどのリスクがある。そこで、本研究では非中央集権的で分散型の電子書籍流通システムを設計した。システムはスマートコントラクトによって電子書籍の所有権を紐つけた NFT を鋳造する販売、その所有権保持者間での P2P ファイル共有などを活用する配信の二つのパートからなる。そして、スマートコントラクトを記述できるプログラミング言語である Solidity やブラウザ間での P2P 接続を可能にする WebRTC などの技術を用いてその試作を行った。また、試作についてダウンロード速度、スケーラビリティの二項目の評価実験を行い、特にスケーラビリティについて課題があることを確認した。

キーワード： Blockchain, Smart Contract, P2P

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　電子書籍の現状

国内だけでも多くの電子書籍ストアがある。利用したことがある人も多いのではないだろうか。多くのストアでは、購入した電子書籍はそのストアの web サイトにアクセス、もしくは専用のアプリにダウンロードすることで閲覧することができる。購入した電子書籍はそのストアのアカウントと紐付けられており、あるストアで購入した電子書籍を他のストアで閲覧するということは基本的にはできない。また、電子書籍を端末にダウンロードした場合にも海賊版対策の観点からコピーに制限がかかっていたり、認証が必要になることがある。電子書籍を購入したといっても購入元の電子書籍ストアがそれを管理しているという点で我々が真に購入しているものはそのストアでの電子書籍へのアクセス権であると言える。

1.2　問題点

電子書籍ストアが抱える問題点を事例を交えながら見ていこうと思う。
まず、電子書籍が電子書籍ストアに管理されていることで以下のようなケースで閲覧できなくなる可能性がある。

1. 書籍の販売が停止された時
2. ユーザアカウントが停止、終了された時
3. サービスが終了した時

書籍の販売が停止された事例を挙げると「Kindle ストア」を運営する Amazon はジョージ・オーウェルによる「1984」と「動物農場」の 2 作品を著作権上の問題で販売停止にする際にユーザの端末上に保存されたデータを遠隔で削除している [1]。ユーザアカウントが停止、削除されるのは不正行為に対する措置というのが第一に考えられる。現に Amazon では転売サイトから Amazon ギフト券を購入しないように呼びかけており、転売もしくは不正に取得された可能性のある Amazon ギフト券の関連するアカウントの停止を行なっている [2]。しかし、他人からアマゾンギフト券をもらった時にそれが転売もしくは不正に取得された可能性がないことを一般のユーザが確認するのは困難であることから、通常の利用のつもりでもユーザアカウントが停止、終了される可能性はある。

サービスが終了した事例を挙げると、「Microsoft Store」が 2019 年に電子書籍の取り扱いを終了し、その際には終了後三ヶ月で電子書籍が読めなくなる代わりに購入金額分を全額返金するという対応がとられた。2014 年にサービス終了した「TSUTAYA.com eBOOKS」では、「BookLive!」への引き継ぎと、引き継げなかったコンテンツに対しては購入金額相当の T ポイントで返金するという対応がとられた。サービス終了時に全く対応を取らなかった例は見当たらなかったが、例えば返金で使われたポイントを普段使わない人や引き継ぎ先のサービスに不満がある人などもいるはずで、全てのユーザが納得する形での撤退というのは現在の電子書籍販売モデルでは難しい。また、ほとんどの電子書籍ストアがクライアント・サーバ型のシステムを採用している。クライアント・サーバ型のデメリットして、中央のサーバへ負荷がかかりやすく、サーバが落ちてしまうとサービス全体が停止してしまう可能性がある。電子書籍ストアはユーザ数が多くても安定して稼働する強力なサーバを用意する必要がある。また、サーバへ情報が集中することにより、ハッキングを受けたときに大きな被害を受ける可能性がある。
また、現在の電子書籍流通では著者がユーザへコンテンツを販売する際に電子書籍ストアを運営する企業が間に入っていることで中間コストが発生し、作家への収益が減っていることも問題としてあげられる。
これらの問題点はいずれも中央集権的な電子書籍流通モデルに起因している。

1.3 既存研究

現在の電子書籍流通モデルの中央集権性を解消する新たなシステムの提案はすでにいくつか存在する。Jeonghee Chi らの研究では、B_Chain と C_Chain の 2 つのブロックチェーンでそれぞれ電子書籍情報と電子書籍購入のトランザクションを管理する安全で信頼性の高い電子書籍取引システムを提案している [3]。株式会社 Gaudiy では、前述の電子書籍サービスが終了した時に閲覧ができなくなる問題への解決策として NFT を活用した電子書籍事業を推進しており、コミックスマート株式会社との共同プロジェクトでは、パブリックブロックチェーンを利用した自立分散型の流通システムを構築、提供するとしている [4]。
これらのシステムで販売においてはブロックチェーンを活用することで非中央集権性を獲得し前述の多くの問題点を解決することに成功しているものの、電子書籍データの配信はサーバに依存しているため、サーバが停止したり、サーバの所有者がその運用をやめた時にシステムが機能しなくなってしまう問題を孕んでいる。

1.4 解決策の提案

本論文では電子書籍流通を販売と配信の二つのパートに分け、販売においてはイーサリアムをはじめとしたパブリックブロックチェーン上で電子書籍の所有権付き NFT の鋳造を行うスマートコントラクトを構築し、配信においては一般的な配信サーバの他に電子書籍の所有権保持者をつなげるハイブリッド P2P 型ファイル共有システ

ムを利用することができる、非中央集権的で分散型の電子書籍流通モデルを提案する。これにより、現在の電子書籍流通モデルが抱える中央集権性を解消しつつ、既存研究の節で言及した配信サーバへの依存を解消する。

1.5 論文の構成

本論文の構成は以下の通りである。第 2 章では提案システムで使用されるブロックチェーンについて述べ、第 3 章では提案システムの試作で使用した WebRTC について述べる。第 4 章では提案システムについて、第 5 章ではその試作を、それぞれ販売パートおよび配信パートに分けて述べる。第 6 章では試作の評価実験について述べる。最後に、第 7 章で本論文の結論を述べる。なお、付録としてコントラクトのコードを加えた。

第2章　ブロックチェーン

2.1　概要

ブロックチェーンとは、暗号技術を用いて情報が格納されたブロックをチェーンのようにつなげていくデータベースであり、その中でも特に P2P 上で管理されているものはパブリックブロックチェーンと呼ばれる。パブリックブロックチェーンの大きな特徴としては耐改竄性、システムが実質的にダウンしないゼロダウンタイムや誰でも簡単にブロックチェーンの中身を見ることができる透明性などが挙げられる。初めに、最も有名なブロックチェーンであるビットコインを例に挙げてブロックチェーンの基本的な仕組みについて解説し、次にイーサリアムとスマートコントラクトについて説明する。

2.2　ビットコイン

ブロックチェーンは 2008 年にサトシ・ナカモトを名乗る匿名の人物または組織によって投稿されたビットコイン論文 [5,6] を発端とする。ビットコインは P2P ネットワーク上で匿名のピア同士が仲介者なしで金銭取引を可能にする電子マネーシステムである。ブロックチェーンは暗号通貨ビットコインの取引を記録する分散型台帳を実現するために利用されている技術であり、暗号通貨ビットコイン自体はブロックチェーンではないことに注意する必要がある。

2.2.1　ブロックチェーン

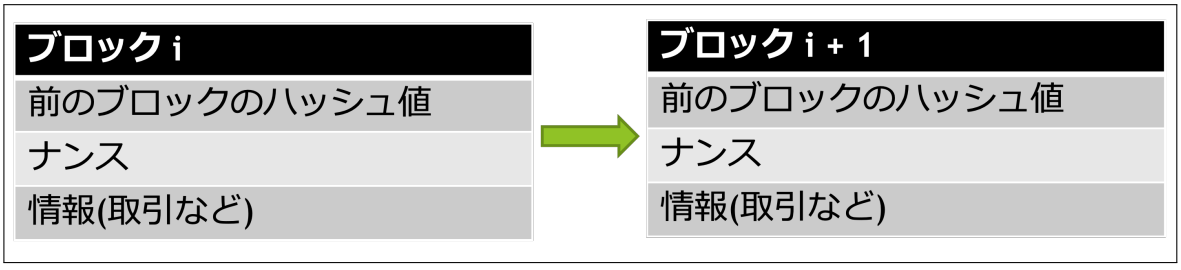

図 2.1: 基本的なブロックチェーンの構造

ブロックチェーンの基本的な構造は図 2.1 のようになっている。ブロックには前のブロックのハッシュ値、ナンス、複数の取引情報などが格納されており、それが 0 番目のブロック (ジェネシスブロックと呼ばれる) から続いている。ハッシュ値はデータをハッシュ化した値であり、ハッシュ化とは元のデータを不規則な値に変換することである。入力が同じ場合は常に同じハッシュ値を返し、入力が少しでも違う場合は全く異なるハッシュ値が返される。そのため、ブロックに含まれた情報が少しでも改竄されればハッシュ値は全く別のものとなり、前のブロックのハッシュ値を記録していくことで繋いだチェーンが切れる事になる。また、ハッシュ値からそのようなハッシュ値を生成するデータを見つけることは非常に難しい。取引情報には送金者により電子署名がかけられており、これによりコインの不正使用を防ぐ。ナンスについては 2.3 で解説する。

2.3 Proof of Work

Proof of work(PoW) は P2P ネットワークでブロックチェーンを維持するための合意形成を行うコンセンサス・アルゴリズムの一つである。ビットコインはブロックチェーンに Proof of work を導入することにより、絶対的とも言える改竄耐性を獲得し、二重支払いを防止することに成功した。

ビットコインの Proof of Work では、新規ブロック生成に先頭に 0 が複数個連続するハッシュ値を生成するナンスと呼ばれる数値を探し出す難しい計算を必要とする。この作業はマイニングと呼ばれ、それを行うノードはマイナーと呼ばれている。ブロックチェーンの改竄を行うと、該当ブロックのハッシュ値が変わる。また、ブロックには前のブロックのハッシュ値が記録されていくため、後続のブロックも書き換える必要がある。従って、改竄地点以降の全ブロックについてマイニング作業を行う必要がある。

P2P ネットワーク内に悪意のあるノードがいて、ブロックに記録された取引を消去し、すでに使ったコインをもう一度使おうと試みるということが考えられる。この時、元のチェーンと改竄されたチェーンの二つのチェーンが存在する。どちらのチェーンが正しいのか、ネットワーク内で多数決をとることで意見を一致させる必要がある。IP アドレス数やビットコインアドレスでの多数決投票では同一人物が複数アドレスを生成するシビル攻撃に弱く、悪意あるノードがいる場合に公平性を担保できない。Proof of work は計算力を票数とした多数決投票としても機能し、より長く成長したチェーンが多数決に勝ったとみなすことができる。長いチェーンほどマイニングを伴うブロック生成が多く行われており、多くの計算力が注がれた証明となるからである。従って、改竄には改竄地点以降の全ブロックについてマイニング作業を行いながら、最も長いチェーンを追い抜く必要がある。また、計算力の小さなグループが計算力の大きなグループのチェーンを追い抜くのはブロックが追加されるごとに指数関数的に難しくなる。そのため、計算力の過半数が良心的なノードによるものである限り、ブロックチェーンを改竄するのは不可能に近い。

マイニングには非常に大きな計算量が必要となるが、ビットコインの場合、条件

を満たすナンスを最も早く見つけたマイナーは新たに発行されたビットコインと取引手数料が報酬として付与される。その報酬を動機とし、現在ではその消費電力が環境問題になるほどマイナーが増えている。

2.4 イーサリアム

イーサリアムはビットコインと同じパブリックブロックチェーンであり、DAppsと呼ばれる分散型アプリケーションを構築するためのプラットフォームである [7]。イーサリアムはグローバルな単一の状態をもち、トランザクションを実行することで誰でもその状態を変化させることができる。ビットコインとの大きな違いとしてチューリング完全のスマートコントラクトを実行できる点が上げられる。これにより、単純な送金処理のみでなく、独自トークンの発行や契約の自動実行ができるようになった。また、ビットコインがUTXO(Unspent Transaction Output)でコインを管理しているのに対してイーサリアムではアカウントベースで取引が管理されている。アカウントにはユーザによって管理される通常のアカウントであるEOA(Externally Owned Account)と、ユーザによってデプロイされたコントラクトに付与されるコントラクトアカウントの二種類がある。EOAは秘密鍵によって管理され、その所有者は暗号通貨イーサリアムの送金やスマートコントラクトの実行を命令するトランザクションを送信することができる。コントラクトアカウントにはコントラクトコードとストレージがあり、コードの実行が命令されればコードの通りに動作し、自身のストレージを操作する。また、他のコントラクトを呼び出すことや、暗号通貨イーサリアムを保持したり、送金することができる。

2.5 スマートコントラクト

スマートコントラクトとは、ブロックチェーン上で実行されるプログラムであり、本稿では特にイーサリアム仮想マシン (EVM) 上でデプロイ、実行されるものを指す。イーサリアムのスマートコントラクトのデプロイ、実行にはGasと呼ばれる仮想燃料をマイナーに支払う必要がある。実行中にGasが切れた場合、プログラムが実行される前の状況に戻る (ただし、消費したGasは返金されない)。この機能により、チューリング完全のスマートコントラクトが無限ループに陥ることを防いでいる。

スマートコントラクトは通常プログラミング言語Solidityなどで記述され、EVMバイトコードにコンパイルされる。コントラクト作成トランザクションを行うことでイーサリアムプラットフォームにデプロイすることができる。スマートコントラクトの大きなメリットは一度デプロイされたコントラクトを書き換えることができないために取引を行うために必要な信用のハードルを下げることができるという点である。例えば、商品を購入する際にお金だけ取られたり異なる商品が渡されたりしないように取引を行うには常に相手を信用する必要がある。しかし、スマートコントラクトはデプロイされた通りに動作し、書き換えられることもないため、誰との取引であろうとトラストレスで行うことができる。スマートコントラクトの応用

例として分散型金融 (DeFi) があり、銀行や政府などの中央有権的な管理者を持たない金融サービスが作られている。DeFi の例としては Uniswap などの分散型取引所やレンディングサービスの Compound などがある。

イーサリアムのスマートコントラクトを使って独自のトークンや NFT を作成することができる。スマートコントラクトを利用するメリットとしては、新規に独自のブロックチェーンやウォレットを作る必要がなく、イーサリアムのセキュリティを利用することができる点などが挙げられる。そうしたトークンにはいくつかの標準が存在するが、そのうち、特に多く使用されている ERC-20 と ERC-721 を紹介する。

2.5.1 ERC-20

ERC-20 は EIP-20 [8] で提案された、代替可能トークンを作る技術標準であり、アドレスに対してトークン保持量を記録する。主要な関数としてはアカウントのトークン保有量を返す balanceOf 関数や別のアカウントにトークンを送る transfer 関数などがある。代表例として、Brave ブラウザで使われている Basic Attension Token(BAT) [9] や、米ドル通貨とほぼ同じ価格を維持するステーブルコインである USDT(Tether) [10] などが挙げられる。

2.5.2 NFT

ERC-721 の紹介の前に、NFT について説明をする。NFT(非代替性トークン) は唯一性をもち、偽造が不可能なデジタルデータであり、絵や動画、ゲームアイテム、その他様々なデジタルコンテンツに結びつけられ、OpenSea をはじめとする NFT マーケットプレイスなどで取引されている。ただし、そのデジタルコンテンツ自体のコピーや不正利用を防ぐものではないことに注意をする必要がある。

2.5.3 ERC-721

ERC-721 は EIP-721 [11] で提案された NFT を作るための技術標準であり、ユニークなトークン ID に対してその所有者のアドレスを記録する。主要な関数としては NFT の所有者を返す ownerOf、別のアカウントに NFT を送る transfer 関数などがある。また、広く使われている拡張として新しい NFT を鋳造する mint 関数を追加する Mintable や NFT にデジタルコンテンツなどのメタデータを結びつける Metadata などがある。

第3章　WebRTC

3.1　概要

WebRTC [12] は主要なウェブブラウザやモバイルアプリケーション間でリアルタイム通信をプラグイン無しで行う機能を追加するオープンソースのプロジェクトであり、ビデオ通話アプリケーションなどに利用されている。ブラウザ間で P2P 通信を行うことでカメラの映像やマイクの音声、その他任意のデータを仲介なしで送受信することができる。WebRTC はデバイスのカメラやマイクへのアクセスを許可するための getUserMedia, ピア同士の接続の管理を行うインターフェイスである RTCPeerConnection, 任意のデータを双方向に送受信するためのインターフェイスである RTCDataChannel などからなる。

3.2　NAT 超え

P2P 通信を行う際にピアが NAT の背後にある場合、多くの場合 NAT 超えと呼ばれる作業が必要となる。NAT はプライベート IP アドレスとグローバル IP アドレスを変換する機器であり、IPv4 の枯渇問題を背景としていたが、現在でも広く使用されている。異なる NAT にいるピア同士で P2P 通信を行うにはピアがグローバル IP アドレスと NAT により割り当てられたポートを知る必要がある。それには、Session Traversal Utilities for NAT(STUN) [13] と呼ばれるクライアントサーバ型のプロトコルが利用される。クライアントが STUN サーバに対してリクエストを送るとき、STUN サーバは IP アドレスとポートを知ることができる、サーバがその情報を返してあげることでクライアントは IP アドレスとポートを知ることができる。STUN を利用しても P2P 通信ができない特定の状況が存在する。その場合 Traversal Using Relays around NAT(TURN) [14] と呼ばれるプロトコルを利用する。TURN では TURN サーバと呼ばれるホストにパケットをリレーさせる。ただし、この場合は P2P とは言えない。Interactive Connectivity Establishment(ICE) [15] は STUN と TURN を利用した UDP ベースの通信用の NAT 超えのプロトコルである。Vanilla ICE 方式では初めに通信経路の候補である ICE Candidate を収集し、そのリストと使用するメディアの情報を Session Description Protocol(SDP) [16] の形式で記述し、相手に送信する。Offer SDP を受け取った相手はそれに対する Answer SDP を記述し相手に送信する。この SDP の送受信にはシグナリングサーバを用意するのが一般的である。

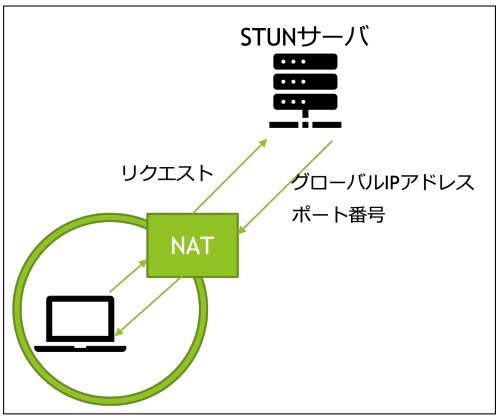

図 3.1: STUN プロトコル説明図

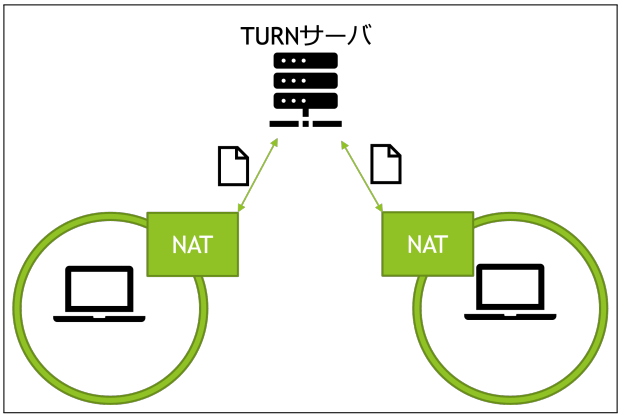

図 3.2: TURN プロトコル説明図

3.3 dataChannel

WebRTC には標準で DTLS によって暗号された任意のデータを送信できる dataChannel という API が用意されている。dataChannel では UDP 上に SCTP が実装されている。SCTP はメッセージ指向型であり、到達保証や順序についてを切り替えることもできる。これを利用して高速なファイル共有アプリケーションを作成することができる。dataChannel.send(message) のようなコードでデータを送信することができ、この message の型としては String や ArrayBuffer、Blob などが利用できる。

第4章　提案システム

4.1　概要

提案システムは販売と配信の二つのパートに分かれている。販売ではスマートコントラクトを作成して作者は自分の作品と価格をスマートコントラクトに登録し、ユーザは所有権付きの NFT を暗号通貨で購入することができる。所有権付きの NFT を入手したユーザは次に、コンテンツのデータそのものを手に入れるため、配信システムを利用してダウンロードを行う。従来の電子書籍流通の配信は電子書籍ストアを運営する企業が用意したもののみが基本であるが、本システムは特定企業に依存しない販売方法をとったことにより、著者が自由に複数の配信方法を用意することができる。本稿では配信手段の一つとして所有権保有者同士を繋ぐハイブリッド P2P 型ファイル共有システムによる配信を提案する。

4.2　販売パート

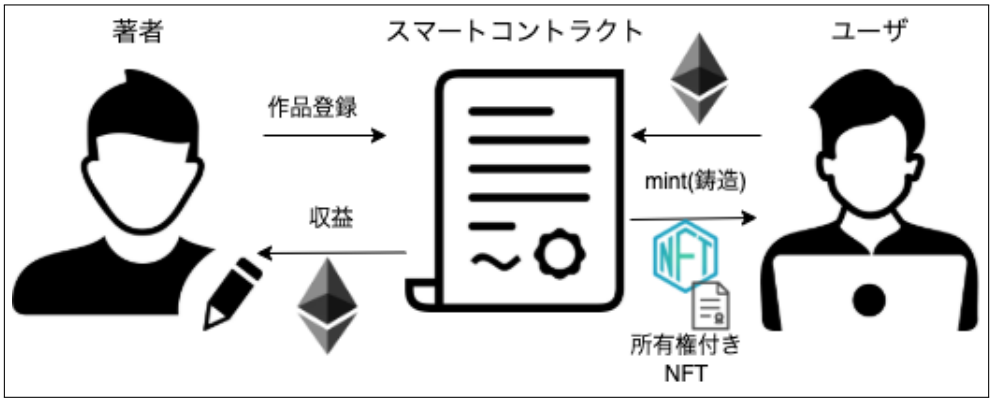

図 4.1: 配信パート概要図

販売パートでは、イーサリアムや EVM 互換性のあるブロックチェーン上に電子書籍の所有権付きの NFT をユーザの暗号通貨と引き換えに鋳造するコントラクトを作成する。電子書籍の著者は自分のアドレスと、作品の価格、ハッシュ値をコントラクトに記録する。登録された電子書籍にはユニークな contentID が付与され、それを用いて著者が後から価格やロイヤリティの設定を変更することも可能である。

ユーザは設定された価格の暗号通貨の支払いを行うと所有権付き NFT を鋳造し、自身のアドレスに付与することができる。

4.2.1 所有権

電子書籍の所有権とはその電子書籍を端末に保持しておく権利であり、提案システムのスマートコントラクトで管理されている。誰でも任意のアドレスのコンテンツの所有権の有無の確認することができる。また、著者は必ず所有権を持っているものとする。著者が所有権を持っていないというのはおかしいという単純な理由からである。重要なアイデアとして、本稿では所有権ホルダー間の電子書籍のデータ共有は違法アップロード、違法ダウンロードにはならないものとする。従って、後述する所有権保有者同士を繋ぐハイブリッド P2P 型ファイル共有システムは著作権上の問題をクリアしているものと考える。

4.2.2 所有権付き NFT

ここでの所有権付き NFT は、より厳密には、所有権を任意のアドレスに付与できる権利をホルダーに与える NFT である。従って、A さんが NFT を持っている時に、B さんに所有権を貸すことができる。また、A さんが NFT を持っている間は任意のタイミングで B さんから C さんに所有権を移したり、A さんが取り戻すことも可能である。つまり、NFT の所有者は借りパクの心配をせず所有権を他人に貸し与えることができる。

4.3 配信パート

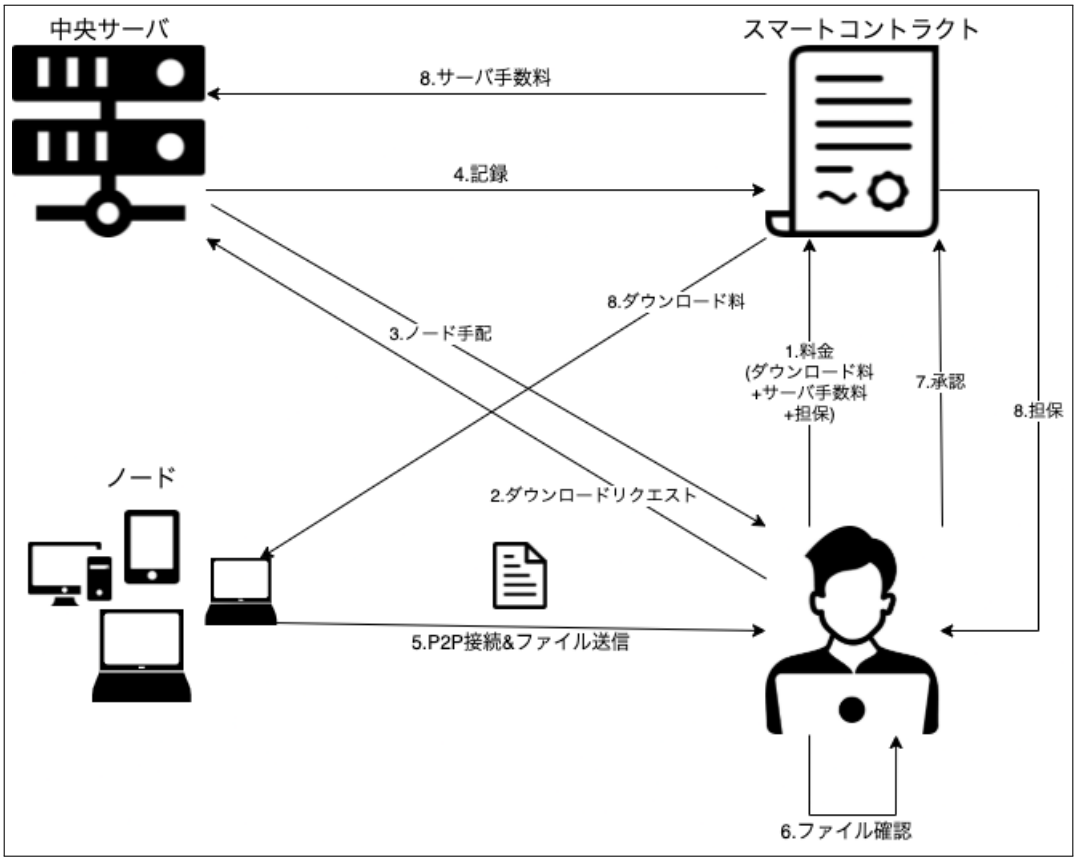

図 4.2: 配信パート概要図

システムは中央サーバ (ブローカー)、不特定多数のノード (配信者)、中央サーバの用意したスマートコントラクトによって構成される。また、ノードおよび中央サーバにはそれぞれダウンロード料と仲介手数料がスマートコントラクトを通してクライアントから支払われるというインセンティブがある。中央サーバは誰もが自由に立てることができ、仲介手数料やノードの選び方なども自由に設定することができる。ノードについても所有権及びコンテンツデータを持っていれば誰でも参加することができる。

4.3.1 スマートコントラクト

スマートコントラクトで担保、仲介手数料、ダウンロード料、残りダウンロードリクエスト回数の管理をする。残りダウンロードリクエスト回数はデフォルトでは 0 であり、クライアントがコンテンツ ID を指定し、ダウンロード料を担保、仲介手数料とともにコントラクトに払うことでそのコンテンツに対する残りダウンロードリクエスト回数を増やすことができる。サーバはクライアントのリクエストしたコ

ンテンツの ID と手配したノードをコントラクトに記録する。クライアントはそのノードからコンテンツデータを受け取り、ハッシュ化してデータが正しいことを確認すると承認を行う。承認と同時に残りダウンロードリクエスト回数は 0 にリセットされ、担保と料金がそれぞれのアドレスに自動で振り込まれる。仮に不良ノードが手配されたりネットワークの不調などで、要求したコンテンツが得られなかった場合、承認する必要はないが残りダウンロードリクエスト回数は減る。そのためコントラクトにダウンロード料を払った時の残りダウンロードリクエスト回数の増加は複数 (例えば 10 など) に設定するべきである。

このコントラクトを用意する代わりに同じような機能を持たせたサーバを用意する、もしくは中央サーバに機能を備えることが可能である。その場合、速度の高速化、ガス代が不要になるなどのメリットがあるが、一方で透明性や信頼性が低下する、単一障害点になるなどのデメリットがある。

4.3.2 中央サーバ

中央サーバは、コンテンツデータを要求するクライアントに対しデータを配信するノードを手配するブローカーとしての役割を果たす。手配する前には販売パートのコントラクトを参照し、クライアントおよびノードが所有権を持っていることと、クライアントの残りダウンロードリクエスト回数が 0 でないことを確認する。手配の際にはクライアントとノードのアドレス、コンテンツ ID をスマートコントラクトに記録する。

4.3.3 ノード (配信者)

ノードは事前に自分が配信することのできるコンテンツを中央サーバに登録し、中央サーバからマッチしたクライアントの情報が伝えられると P2P 接続を行いコンテンツデータをクライアントに送信する。

4.4 メリット

提案システムには複数のメリットが存在する。まず、海賊版対策におけるメリットが挙げられる。今日の電子書籍のシステムではユーザが購入、利用した電子書籍の権利を確認することは難しい。そのため、犯罪者が違法に電子書籍をコピーし、営利目的で販売、配信しても、一般のユーザからはそれが違法なのかを判断することができず、知らないうちに海賊版の被害者になっている可能性がある。ブロックチェーンを利用することで権利が明確になり、またそれを容易に確認することや他人に譲渡したりすることが可能になる [17]。これにより、海賊版ユーザの摘発や抑制つながることが予想される。

次に、NFT が中古で売られた時にロイヤリティを設定することができる点が挙げられる。NFT のロイヤリティの標準は EIP-2981 [18] で提案されており、本システ

ムでもこれを取り入れることができる。ロイヤリティを設定することで著者の収入をより継続的なものにすることができる。

最後に、本システムは分散型であることから検閲への耐性を持っており、提案システムのユーザはいかなる国に住んでいても書籍の出版、購入が可能となる。

第5章　試作

5.1　販売

表 5.1: Token 構造体の変数とその説明

変数	説明
contentId	トークンと紐ついているコンテンツの ID
addressOwnershipGranted	NFT がコンテンツ所有権を与えているアドレス

表 5.2: Content 構造体の変数とその説明

変数	説明
author	著者のアドレス
price	NFT を発行する時に必要な価格
hash	コンテンツのハッシュ値
ipfsPath	ipfs に保存したメタデータへのパス
ownerships	アドレスに対し所有権の数を返す Mapping

Solidity 言語とイーサリアムで動作するスマートコントラクトの開発を補助するツールである Hardhat を使用し、所有権付き NFT を鋳造するコントラクトを作成した。また、ERC2981 のロイヤリティや IPFS を用いたメタデータの保存についても対応させた。このコントラクトは openzeppelin により提供されている ERC721.sol[1] を継承している。図のように Token および Contract の二つの構造体を定義する。また、トークン ID から該当するトークンを返す_tokens とコンテンツ ID から該当するコンテンツを返す_contents の二つのマッピングを宣言する。トークン ID、コンテンツ ID をユニークなものにするため、nextContentId、nextTokenId で次に作られる ID を管理する。主要な関数として

[1] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

mint

新しいコンテンツ所有権付き NFT を鋳造する関数である。コンテンツ ID とアドレスを引数にとる。初めにコンテンツ ID が存在していることを確認する。次に実行者が author でない場合、設定された値段に基づき実行者から author へ暗号通貨の支払いが行われる。支払いに成功すると ERC721.sol の_safeMint 関数を呼び出し、nextTokenId をトークン ID とした NFT を新たに鋳造する。また、_tokens マッピングで nextTokenId がキーの要素の contentId に引数のコンテンツ ID を代入する最後に、nextTokenId をインクリメントする。

register

作品を登録するための関数である。価格とハッシュ値、ロイヤリティ、ロイヤリティ受取人のアドレス、ハッシュ値、ipfs のパスを引数にとる。_contents マッピングで nextContentId がキーの要素にそれぞれ値を代入し、nextContentId をインクリメントする。また、価格とハッシュ値のみを引数にとる同名関数も用意した。

hasOwnership

アドレスがコンテンツの所有権を保持しているかを確認するための関数である。アドレスとコンテンツ ID を引数にとり、そのアドレスがそのコンテンツの所有権を持っているかを Content 構造体の ownerships マッピングを参照し、bool 値で返す。また、アドレスがそのコンテンツの author のものでも true を返す。この関数はデータの保存や変更を伴わない view 関数なので実行にガス代はかからない。

_beforeTokenTransfer

この関数は内部関数であり、ERC 721.sol の同名関数の override である。その名前の通り、トークンの移動の際に呼び出され、所有権の移動を行う。この関数はトークンの移動元のアドレスである from, 移動先のアドレスである to、そしてトークン ID を引数とする。指定されたトークンのコンテンツにおいて ownerships の from キーの要素をデクリメントし、to キーの要素をインクリメントする。また、トークンの addressOwnershipGranted を to に変更する。ERC721.sol の_mint 関数内でもゼロアドレスからの送付先アドレスへのトークンの移動としてこの関数が呼び出されている。

が存在し、そのほかにも一部の情報だけを変更するための set 関数やマッピングの値を返す view 関数を複数実装した。

5.2 配信

5.2.1 概要

スマートコントラクトを販売パートと同じ環境で実装、Node.js で中央サーバを実装し、WebRTC を使ったデータ送信およびスマートコントラクトを実行する GUI

としての機能を持つ web アプリケーションを React.js, ethers.js などのライブラリを使用して作成した。STUN サーバおよび、TURN サーバについては SkyWay により無料で提供されているものを使用した。また、全てのコードは javaScript ではなく TypeScript で記述している。

5.2.2 スマートコントラクト

ダウンロードはコンテンツ ID に対し、ダウンロード料を返す_downloadFees mapping で管理する。ダウンロード料は著者が自由に設定、変更することができる。その他、クライアントのアドレスとコンテンツ ID に対し残りダウンロードリクエスト回数を返す_count、払ったダウンロード料を返す_paidDlFee、払ったサーバ手数料を返す_paidBrokerageFee、手配されたノードのアドレスを返す_arrangedNode の mapping がある。

主要な関数として

payDownloadFee 関数

クライアントはこの関数でダウンロードしたいコンテンツの ID を指定し、担保とサーバ手数料、ダウンロード料を支払う。担保はダウンロード料と同じ金額である。支払いに成功すると指定したコンテンツの残りダウンロード回数が 10 増える。_paidDlFee に支払ったダウンロード料を、_paidDlFee にサーバ手数料を記録する。

setArrangedNode

中央サーバのみが実行できる関数で、クライアントのアドレス、コンテンツ ID、ノードのアドレスを記録するとともに、クライアントの残りダウンロード回数を 1 減らす。

approveNode

クライアントは setArrangedNode 関数で記録されたノードからコンテンツデータを正しく受け取った場合この関数を実行する。実行すると支払った担保が返金され、残りのダウンロード料がノードに、サーバ手数料がサーバにそれぞれ支払われる。また、クライアントのそのコンテンツ ID に対する残りダウンロード回数、_paidDlFee、_paidBrokerageFee を 0 にする。

を用意した。

5.2.3 中央サーバ

システムを利用する全ユーザには初めに Socket.IO を使用して中央サーバとコネクションを確立させる。その後、socket.id をメッセージとした電子署名を要求し、それを復号することでユーザのイーサリアムアドレスを決定する。ユーザが希望するロールがノードであれば Node クラスのインスタンス、クライアントであれば

表 5.3: Node クラスの変数とその説明

変数	説明
socket	ノードの socket
account	イーサリアムアドレス
client	手配しているクライアント。手配していない場合は null
contentIds	ノードが提供できるコンテンツの ID の集合
offerSDP	ノードの offerSDP

表 5.4: Client クラスの変数とその説明

変数	説明
socket	クライアントの socket
account	イーサリアムアドレス
contentId	要求中のコンテンツ ID
node	手配されたノード
answerSDP	ノードの offerSDP に対する answerSDP

Client クラスのインスタンスを作り出し、socket のイベントも新たに追加する。追加されるイベントは、具体的には、ノードの場合配信するコンテンツ ID を設定する setContent イベント、offerSDP を設定する setOfferSDP、クライアントへの配信が完了したときに送る finish イベントなどがある。クライアントの場合は setAnswerSDP や requestContent イベントなどがある。クライアントは requestContent イベントを利用してコンテンツを要求することができ、サーバはクライアントのコンテンツ所有権および残りダウンロードリクエスト回数を確認し、ノードを選択し、ノードの了解を得られれば offerSDP をクライアントに返す。ノードは setContent イベントを利用してサーバにコンテンツ ID を伝えるとコンテンツ所有権を持っていることを確認したのちにそのコンテンツを配信できるノードを管理する Set に追加され、クライアントによる requestContent 実行時にランダムに選ばれるノードの候補になることができる。逆に disconnect イベントが起きた時には候補から削除される。

5.2.4 フロントエンド

イーサリアム系ブロックチェーンとのやりとりをするためにユーザは事前にブラウザに chrome 拡張機能として提供されている仮想通貨ウォレット Metamask をインストールしておく必要がある。Web アプリケーションを開くと、「MetaMask を使用して接続」というタブが開かれるので次へを押して接続する。ノードとクライアントのどちらのロールをするかを選択し、socketID のメッセージで署名を行う。ノー

ドの場合配信したいコンテンツのファイルを読み込み、そのコンテンツ ID を送信し待機する。クライアントの場合は欲しいコンテンツの ID を入力しリクエストを行うと、サーバによりノードが手配される。ノードとクライアント間で SDP を交換し、WebRTC の接続を確立することができたらノードからクライアントへのコンテンツデータの送信が開始される。ノードは事前にコンテンツデータを ArrayBuffer の形式で読み込んであり、dataChannel を通して約 64KB にスライスしながら送信する。受信が完了したらスライスされたデータを繋ぎ合わせ、SHA-256 でハッシュ値を計算する。ハッシュ値が正しければノードとの接続を断つ。切断されたノードは中央サーバにそれを伝え、再びクライアントが手配されるまで待機する。クライアントは後からコントラクトの approveNode 関数を実行することで担保を回収することができる。

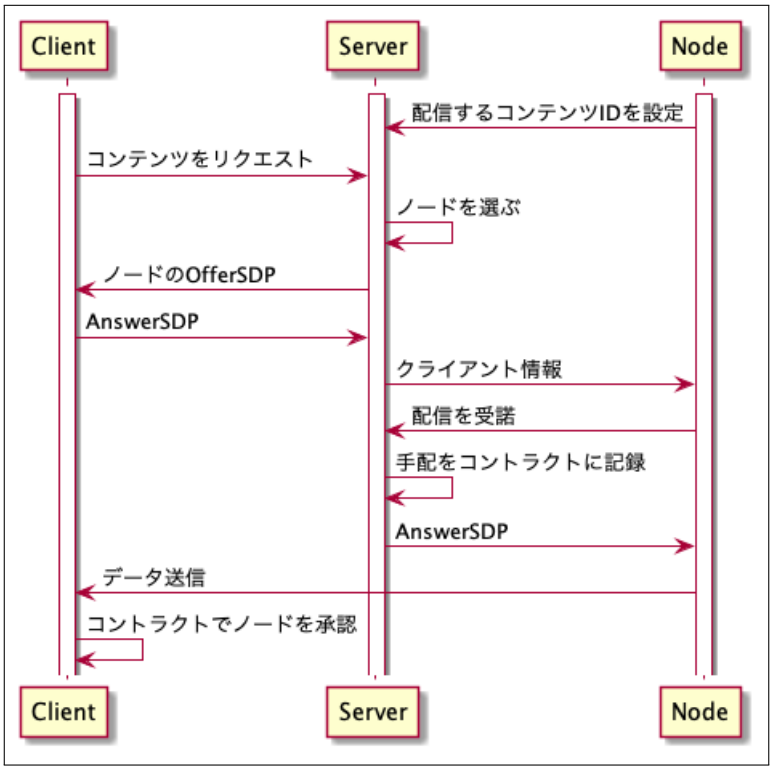

図 5.1: クライアントがコンテンツを受け取るまでのフロー

第6章　評価実験

6.1　概要

本章では、特に配信システムの評価実験について述べる。6.2 では、データサイズを変化させながらクライアントがリクエストをしてからダウンロードが完了するまでの時間を調べた。6.3、及び 6.4 ではユーザが増えた場合にどのようにシステムに影響があるかを調べるために Socket.IO の同時接続数とクライアント登録のレスポンスの変化を調べた。

6.2　ダウンロード速度

クライアントがコンテンツデータのリクエストしてからダウンロードするまでの一連の流れにかかる時間についてを計測した。

6.2.1　実験方法

データをリクエストしてノードの OfferSDP が帰ってくる時間、そこから WebRTC が接続を確立し、クライアント側で dataChannel.onOpen イベントが発火するまでの時間、データの受信にかかる時間、ハッシュ化にかかる時間を 1MB、10MB、100MB、1GB のデータについて計測した。また、中央サーバを heroku を利用して構築し、Mumbai Testnet にコントラクトをデプロイした。wi-fi に接続された MacBook をノードとし、有線 LAN 接続されたデスクトップ PC をクライアントとして利用した。

6.2.2　結果

当然ではあるが、データサイズが大きくなるとそれに比例してデータ受信、ハッシュ化にかかる時間が増えていることがわかる。特に、100MB 以上の場合、データ受信にかかる時間が大きな割合を占めている。また、電子書籍のデータサイズはおよそ漫画が 40MB 150MB、小説が 1MB 20MB、雑誌・写真集が 50MB 300MB である [19]。漫画や雑誌、写真集を扱う場合はデータ受信の時間をいかに短縮できるかが特に重要であると言える。

size \ s	全体	OfferSDP 受信	WebRTC 接続	データ受信	ハッシュ化
1MB	1.58	0.25	1.14	0.15	0.03
10MB	2.72	0.27	1.07	1.20	0.19
100MB	14.92	0.27	1.0	11.9	1.75
1GB	141.57	0.27	1.0	122.63	17.64

表 6.1: リクエストからの時間

6.3 同時接続数

本システムではノードの多くは長時間待機していると考えられるため、大きな同時接続数を予想することができる。そこで、Socket.IO でどの程度の同時接続に耐えることができるのかを調べる。

6.3.1 実験方法

デバイス名	HP ZBook Fury 17.3 inch G8 Mobile Workstation PC
OS	windows 10 Pro
プロセッサ	インテル ® Core™ i7-11850H プロセッサー
RAM	32.0 GB

表 6.2: ZBook Fury のスペック

HP のモバイルワークステーションである ZBook Fury 上で実験を行った。VMware Workstation を使用して Ubuntu 21.10 の仮想マシンを作成し、その上に中央サーバを立てた。テストツールには Artillery を使用し、ホスト OS から仮想ユーザを複数作成する。仮想マシンに割り当てるプロセッサ数やメモリを変化させながら、Socket.IO の接続をしてそのまま待機する仮想ユーザを秒間 50 人、4 分間の計 12000 人作成し、同時接続数のテストを行った。

6.3.2 同時接続数のテスト結果

プロセッサ数が 1 のとき、割り当てメモリが 2GB の場合 5993 人のユーザが、4GB の場合 5990 人の仮想ユーザがエラーなく動作を完了することに成功した。プロセッサ数が 2 のときは、メモリが 2GB の場合 6641 人、4GB では 10049 人であった。また、これより大きなリソースを割り当てた時、作成した仮想ユーザが 12000 人に満たない状態でテストが終了するということが見られた。理由としては、ホスト OS 側のリソースの不足だと思われる。

少なくとも 5000 程度の同時接続数には耐えられることがわかる。また、プロセッサ数 2、メモリ 2GB の場合、10000 人を超えていることからリソースを増やすことで同時接続数を増やすことができたと言える。

	成功ユーザ数	xhr post エラー	xhr poll エラー
1 プロセッサ、2GB	5993	233	5774
1 プロセッサ、4GB	5990	562	5448
2 プロセッサ, 2GB	6641	638	4507
2 プロセッサ, 4GB	10049	826	1125

表 6.3: Socket.IO の同時接続数に対するテスト

6.4 レスポンス速度の変化

特に、人気漫画の販売開始直後などはクライアントが殺到する。そのような場合にどの程度であればレスポンスに影響がないかを調べたい。

6.4.1 実験方法

ZBook Fury を再び使用する。仮想マシンは、プロセッサ数を 2、メモリを 2GB 割り当てた。同じく Artillery を使用し、Socket.IO の接続後、クライアントとしての登録を行う仮想ユーザを作成し、秒間クライアント登録数の変化に伴うレスポンスの変化をテストした。

6.4.2 レスポンス速度の変化のテスト結果

秒間クライアント登録数が 100 以下では 50ms 付近で横ばいになっているが、それ以降は値が大きくなるほどレスポンスに遅れが生じてきていることがわかる。秒間クライアント登録数が 150 の時にはグラフ中最大の 837.3ms を記録している。また、秒間仮想ユーザ数を 160 以上にするとエラーが発生してしまい、うまくテストを完了することができないという事象が頻発した。

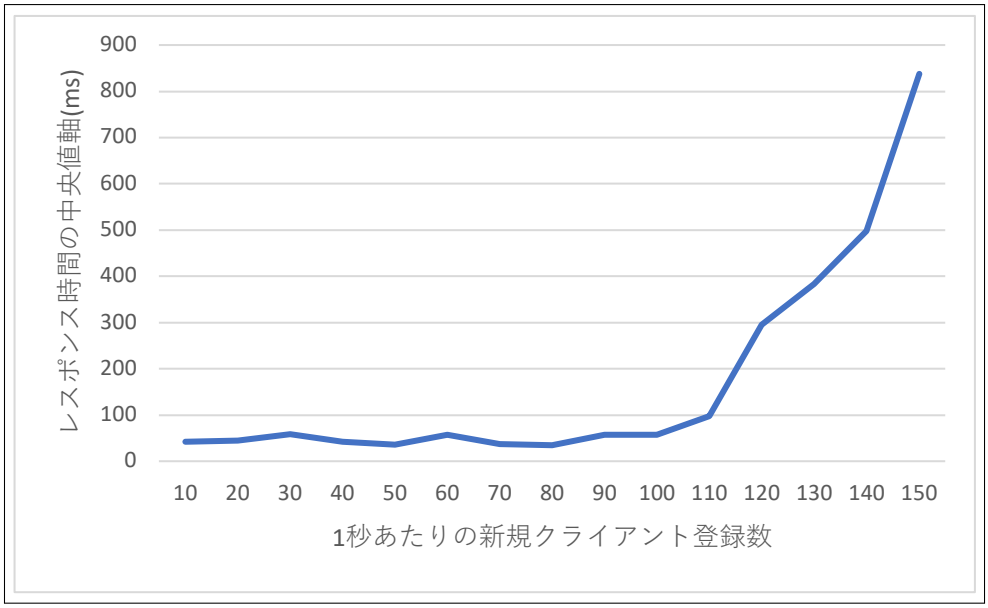

図 6.1: 秒間クライアント登録数とレスポンス時間の中央値

第7章　結論

7.1　まとめ

今日の電子書籍流通システムには中央集権的であるが故にストアの終了などにより購入した本が読めなくなるリスクなどが存在する。また、クライアント・サーバ方式が基本であることから単一障害点の存在やハッキング被害のリスクが存在する。本論文ではそれらの問題の解決するべく非中央集権的、分散型の流通システムを目指した。電子書籍の所有権を紐つけた NFT を鋳造するスマートコントラクトを用いた販売と、配信方法の一つとして所有権保有者間での P2P ファイル共有を利用できる流通システムを設計し、スマートコントラクトを記述するためのプログラミング言語である Solidity や Socket.IO, WebRTC などのリアルタイム通信を支えるライブラリなどを使いその試作、またそれに対する評価実験を行った。

7.2　課題

販売においては、実際には作者ではない人が作品を登録するなど、詐欺への対策が課題である。これは本システムのみならず、ユーザが自由に NFT を発行できるサービス全てに言えることである。これの単純な解決方法はユーザが直接 NFT を鋳造するのではなく、信頼できる組織などを間において、ユーザが作品の本物の作者であることを確認したのちに鋳造するなどが挙げられる。一方で、この方法では非中央集権性が失われる。

本研究で試作した配信システムにおいては、ユーザ数が多い場合、エラーの発生で Socket.IO の接続を確立できなくなったりレスポンスに遅れが生じることがわかった。実運用を考えると、人気漫画の販売開始直後などはユーザ数が急激に増加することなどが予想される。そういったケースにも対応できるよう、スケーラビリティを向上させる必要がある。また、中央サーバを必要とするハイブリッド型 P2P であり、より分散性の高い配信方法も考えたい。

また、ブロックチェーン自体もスケーラビリティを課題としており、高速なチェーンを用いる、できる限りオフチェーンで情報を処理するなどの工夫が必要である。

謝辞

コロナ禍という難しい状況の中で本研究の遂行および本論文の執筆にご支援、ご指導いただきました藤田聡教授に深く感謝を申し上げます。

参考文献

[1] Amazon Erases Orwell Books From Kindle
https://www.nytimes.com/2009/07/18/technology/companies/18amazon.html

[2] Amazon.co.jp: ご購入に関する注意: ギフト券
https://www.amazon.co.jp/b?ie=UTF8&node=4705466051

[3] Chi J, Lee J, Kim N, Choi J, Park S (2020) Secure and reliable blockchain-based eBook transaction system for self-published eBook trading. PLoS ONE 15(2): e0228418. https://doi.org/10.1371/journal.pone.0228418

[4] 【プロジェクト#3】NFT × 電子書籍 — 株式会社 Gaudiy
https://hp.gaudiy.com/medias/364/

[5] Satoshi Nakamoto(2008) Bitcoin: A Peer-to-Peer Electronic Cash System https://bitcoin.org/bitcoin.pdf

[6] Bitcoin - Open source P2P money https://bitcoin.org/

[7] Ethereum https://ethereum.org/en/

[8] Fabian Vogelsteller, Vitalik Buterin(2015), EIP-20: Token Standard https://eips.ethereum.org/EIPS/eip-20

[9] Basic Attension Token https://basicattentiontoken.org/

[10] Tether https://tether.to/

[11] William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs(2018), EIP-721: Non-Fungible Token Standard https://eips.ethereum.org/EIPS/eip-721

[12] WebRTC https://webrtc.org/

[13] M. Petit-Huguenin, Impedance Mismatch, G. Salgueiro, Cisco, J. Rosenberg, Five9, D. Wing, Citrix, R. Mahy, Unaffiliated, P. Matthews, Nokia(2020) rfc8489 https://datatracker.ietf.org/doc/html/rfc8489

[14] T. Reddy, Ed., McAfee, A. Johnston, Ed., Villanova University, P. Matthews, Alcatel-Lucent, J. Rosenberg, jdrosen.net(2020) rfc8656 https://datatracker.ietf.org/doc/html/rfc8656

[15] A. Keranen, C. Holmberg, Ericsson, J. Rosenberg, jdrosen.net(2018) rfc8445 https://datatracker.ietf.org/doc/html/rfc8445

[16] A. Begen, Networked Media, P. Kyzivat, C. Perkins, University of Glasgow, M. Handley, UCL(2021) rfc8866 https://datatracker.ietf.org/doc/html/rfc8866

[17] Zhengjun Cao, Zhen Chen, Ruizhong Wei, Lihua Liu (2018) A Survey of E-book Digital Right Management

[18] Zach Burks, James Morgan, Blaine Malone, James Seibel(2020) EIP-2981: NFT Royalty Standard https://eips.ethereum.org/EIPS/eip-2981

[19] 電子書籍の容量は何 GB 必要？本の種類とファイルサイズから考える最適解 https://www.kyodotokyo.com/book-movie/ebook-required-capacity/

付 録A　コントラクトのコード

A.1　購入パート

```
function mint(
        uint256 contentId,
        address to
    ) payable external {
        require(contentId < nextContentId, "content not existed");

        if(msg.sender != _contents[contentId].author) {
            require(_contents[contentId].price == msg.value, "msg.
                value not equal the price");
            payable(_contents[contentId].author).transfer(msg.value
                );
        }

        _tokens[nextTokenId].contentId = contentId;

        _safeMint(to, nextTokenId, '');

        nextTokenId += 1;
    }
```

Listing A.1: mint 関数

```
function register(uint256 price, uint256 royalty, address receiver,
    string memory hash, string memory path) public {
        _contents[nextContentId].author = msg.sender;
        _contents[nextContentId].price = price;
        _contents[nextContentId].royalty = royalty;
        _contents[nextContentId].receiver = receiver;
        _contents[nextContentId].hash = hash;
        _contents[nextContentId].ipfsPath = path;

        nextContentId += 1;
    }
```

Listing A.2: register 関数

```
function hasOwnership(address account, uint256 contentId) public
   view returns(bool) {
        return _contents[contentId].ownerships[account] != 0 ||
            _contents[contentId].author == account;
    }
```

Listing A.3: hasOwnership 関数

```
function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal override {
        if(from != address(0)) {
            _contents[_tokens[tokenId].contentId].ownerships[from]
               -= 1;
        }
        if(to != address(0)) {
            _tokens[tokenId].addressOwnershipGranted = to;
            _contents[_tokens[tokenId].contentId].ownerships[to] +=
                1;
        }
    }
```

Listing A.4: _beforeTokenTransfer 関数

A.2 配信パート

```
function payDownloadFee(address client, uint256 contentId) public {
        require(owt.hasOwnership(client, contentId), "you don't
           have the ownership");
        require(_downloadFees[contentId] != 0, "download fee should
            not be zero");
        require(_remainingCount[client][contentId] == 0, "still
           have chances");

        uint256 brokerFee = _downloadFees[contentId] *
           brokerageFeeRate / 100;
        fst.transferFrom(msg.sender, address(this), _downloadFees[
           contentId] * 2 + brokerFee);

        _paidDlFee[client][contentId] = _downloadFees[contentId];
        _paidBrokerageFee[client][contentId] = brokerFee;

        _remainingCount[client][contentId] = _downloadLimit;
        if(_arrangedNode[client][contentId] != address(0)) {
            _arrangedNode[client][contentId] = address(0);
        }
    }
```

Listing A.5: payDownloadFee 関数

```
function setArrangedNode(address client, uint256 contentId, address
    node) external {
        require(msg.sender == _server);
        require(owt.hasOwnership(client, contentId));
        require(owt.hasOwnership(node, contentId));
        require(_remainingCount[client][contentId] >= 1, "you need
           to pay download Fee");
        _remainingCount[client][contentId] -= 1;
        _arrangedNode[client][contentId] = node;
```

```
8       }
```

Listing A.6: setArrangedNode 関数

```
1  function approveNode(address client, uint256 contentId) external {
2          require(msg.sender == client, "You don't have permission.")
               ;
3          require(_arrangedNode[client][contentId] != address(0), "
               node not arranged");
4          address node = _arrangedNode[client][contentId];
5          _arrangedNode[client][contentId] = address(0);
6
7          fst.transfer(node, _paidDlFee[client][contentId]);
8          fst.transfer(client, _paidDlFee[client][contentId]);
9          fst.transfer(_server, _paidBrokerageFee[client][contentId])
               ;
10         _paidDlFee[client][contentId] = 0;
11         _paidBrokerageFee[client][contentId] = 0;
12
13         if(_remainingCount[client][contentId] != 0) {
14             _remainingCount[client][contentId] = 0;
15         }
16     }
```

Listing A.7: setArrangedNode 関数