ブロックチェーンを活用した IoT 機器のソフトウェア更新フレームワーク

中西遼太[1]　松原豊[1]　高田広章[1]

概要： IoT 機器の急増に際して，IoT 機器を対象とするサイバー攻撃も増加している．IoT 機器を対象とするサイバー攻撃を防ぐために，機器や機器に搭載されるソフトウェアの開発者（機器開発者）には，IoT 機器に含まれる脆弱性を修正するためのソフトウェアを迅速に更新することが求められている．機器管理者だけでなく，機器を購入して利用する者（機器利用者）の負担を低減するために，IoT 機器向けソフトウェアの配布・更新機能は，低コストで容易に実現でき，かつ人間の関与をできる限り少なくなるよう一連の処理を自動化したフレームワークとして開発することが望ましい．加えて，ソフトウェア更新フレームワークには，ソフトウェアの配布・更新機能を悪用したサイバー攻撃へ耐性も期待される．本論文では，ブロックチェーンとスマートコントラクトによる自動化を特徴とする，IoT 機器向けソフトウェア更新フレームワークを提案する．まず，機器管理者や機器利用者などのステークホルダと，そのユースケースを分析し，提案フレームワークの要件を整理する．その結果にもとづいて，フレームワークの設計及び実装について述べる．評価では，既存の関連研究との定性的及び定量的に比較し，提案フレームワークによって，ステークホルダの負担を減らせる可能性があることを示す．

1. はじめに

近年，Internet of Things（IoT）が注目され，ネットワークに接続される IoT 機器が増加している．その数は，2018 年時点で 307 億台に達している[1]．今後も，IoT を活用した新しいサービスが登場，発展するとともに，IoT 機器の数も増加すると考えられる．その一方で，IoT 機器を対象とするサイバー攻撃も増加している[2]．IoT 機器が製品として出荷された後に，新たに脆弱性が発見され，攻撃者がその脆弱性を利用すると，機器利用者のプライバシーや安全性の侵害に繋がる可能性がある．これらをできる限り防止するために，機器開発者には，IoT 機器に含まれる脆弱性を修正するためのソフトウェアを迅速に開発・配布し，対象機器のソフトウェアを更新することが求められる．しかしながら，現状では，ソフトウェア更新機能が提供されていない機器も少なからず存在しており，出荷した機器に脆弱性が発覚すると，機器開発者が莫大なコストを掛けて機器を回収して修正[3]することや，最悪の場合は，脆弱性を修正せずに放置しまうこともある．最近では，出荷後のソフトウェア更新に備えて，IoT 機器を（家庭内 Wi-Fi ルータやスマートフォンを介して）インターネットに接続して遠隔からソフトウェアを更新する Over-the-air（OTA）と呼ばれる機能を搭載する機器も増えている．

ソフトウェア更新機能をもつ IoT 機器を開発する際には，機器開発者の責任のもとで，ソフトウェアを開発，配布する仕組みも併せて提供・運用することになり，機器開発者の負担が増加する．一方，ソフトウェア更新機能の実現方法は製品によって大きく異なるので，複数の IoT 機器を使用する機器利用者の立場からすると，自身が保有する多種多様な機器を管理し，それぞれのセキュリティを適切に保つためにソフトウェア更新を継続することは容易ではない[4]．

そこで，機器開発者と機器管理者の両方の負担を低減するために，IoT 機器向けソフトウェアの配布・更新機能を低コストで容易に実現でき，かつ人間の関与をできる限り少なくなるよう一連の処理を自動化したフレームワークの研究が進められている[4-8]．しかしながら，実用を想定した場合，先行研究では，ソフトウェア更新のアルゴリズムやフレームワークの設計にとどまっており，実際に実装して評価をしている研究がほとんどない．

本研究では，ブロックチェーンとスマートコントラクトによる自動化を特徴とする，IoT 機器向けソフトウェア更新フレームワークを提案する．まず，ステークホルダ（機器管理者と機器利用者）と，そのユースケースを分析し，提案フレームワークの要件を整理する．その結果にもとづいて，次の特徴を有するフレームワークの設計及び実装について述べる．

1. 機器開発者は，フレームワークに登録された全機器に，同時にソフトウェアを配布し，更新状態を把握できる（ソフトウェア配布・更新の自動化）
2. 機器管理者は，IoT 機器をフレームワークに登録すれば，自動的にソフトウェアを更新でき（ソフトウェア更新の自動化），機器のソフトウェアのバージョンと更新履歴を把握できる（機器管理の容易化）

評価では，フレームワーク構造，ステークホルダの負担，セキュリティの観点で，既存研究と定性的に比較し，提案フレームワークによって，ステークホルダの負担を減らせる可能性があることを示す．さらに，ソフトウェア更新機能の実行時間と IoT 機器におけるメモリ消費量の観点で，定量的に評価し，提案フレームワークの実用性を示す．

1 名古屋大学大学院情報学研究科
Graduate School, Nagoya University

本研究の貢献は，以下の通りである．

3. IoT 機器向けソフトウェア更新フレームワークの実用を想定した，ユースケースと要件を示したこと．
4. 要件にもとづいて，提案フレームワークの具体的な設計と実装の事例を示し，実現可能性を示したこと．
5. 実装した提案フレームワークを定性的及び定量的に評価し，実用の可能性を示したこと．

本論文の構成は，以下の通りである．第 2 章では関連研究として，ブロックチェーンについて説明し，先行研究についても紹介する．第 3 章では，提案フレームワークの設計と実装について述べる．第 4 章では，既存研究との比較を通じて，提案フレームワークを評価する．最後に，第 5 章で，本論文のまとめと今後の課題について述べる．

2. 関連研究

2.1 ブロックチェーン

ブロックチェーンは，複数の参加者（コンピュータ）で分散管理される台帳で，コンセンサスアルゴリズムによって分散台帳の整合性を確保する．コンセンサスとは，ブロックチェーン上の取引（トランザクション）を，アルゴリズムに基づいて参加者が検証し，妥当な取引のみを台帳であるブロックチェーンに記録することである．検証されたトランザクションは，複数集めてブロックとして格納される．各ブロックには，トランザクションに加えて，1 つ前に生成されたブロックのハッシュ値を格納する．複数のブロックをハッシュ値によりチェーン状で構成することからブロックチェーンと呼ばれる．ブロックチェーン上のあるブロックに格納された取引を改竄しようとした場合，対象としたブロックの後続のブロックのハッシュ値と，改竄したブロックから計算されるハッシュ値が異なるので，ブロックチェーンの検証によって容易に検出可能である．改竄したブロックに後続する全てのブロックを改竄するのは，（理論的には可能であるが）現実的には困難であるという状況を維持することで，外部からの攻撃や障害に強いインフラを低い実装コストで構築できるとされる[9].

ブロックチェーンには大きく分けて 2 種類の形態が存在し，パブリック型チェーンとパーミッション型チェーンに分けられる．ブロックチェーンを利用した技術として代表される仮想通貨 Bitcoin[10]はパブリック型チェーンとして不特定多数の参加者が存在する．それに比べて Hyperledger[11]に代表されるパーミッション型チェーンは，特定の企業・組織のみが参加できるブロックチェーンであり，参加するのにブロックチェーンの運営者に許可をとる必要がある．パブリック型とパーミッション型ではコンセンサスの取り方が大きく異なる．

Bitcoin に代表されるパブリック型チェーンは不特定多数の参加者が存在する．その為，悪意のある参加者も存在する可能性がある．これを防ぐ為に，多くのパブリックチェーンでは，参加者に膨大な計算を行わせるマイニングを導入している．これはハッシュの逆算といった計算量が膨大な処理を参加者に実行してもらい，最初に解となる値を導出した参加者に対してトランザクションの検証及びブロックに追加する権利を報酬とともに与える仕組みである．このような膨大な計算処理を経てブロックに追加する作業を Proof of Work（PoW）と呼び，ブロックの改竄を防ぐ目的として導入している．

Hyperledger に代表されるパーミッション型チェーンはさらに 2 つの分類が可能であり，コンソシーアム型チェーンとプライベート型チェーンに分けることが可能である．プライベート型チェーンは 1 つの企業・団体が，コンソシーアム型チェーンは複数の企業・団体で管理・運営されているチェーンであり，参加するには許可をとる必要がある．信頼できる参加者のみで構成されることから，PoW よりも軽量なコンセンサス方式をとることが可能であり，台帳の更新といったトランザクションの確定までの時間がパブリック型に比べて迅速であることがメリットである．

スマートコントラクトは，Szabo によって提唱[12]されており，現実世界の契約を計算機ハードウェアもしくはソフトウェアとして実装する手法について述べられている．ブロックチェーンにおけるスマートコントラクトは，ブロックチェーン上で動作・実行可能なアプリケーションを指す．仮想通貨などの資産や権利がブロックチェーンに記録されると，それらを使った取引や権利の使用に関してスマートコントラクトを通じて自動化することが可能になっている．ブロックチェーン上のスマートコントラクトが実行されると，実行結果がブロックチェーンに記録される為，取引結果や契約の執行を第三者機関なしで保証することが可能である．

文献[13]において，IoT におけるブロックチェーンの活用例が多く述べられており，急増する IoT 機器の管理もその 1 つになっている．本研究でも，IoT 機器の管理とソフト更新機能を自動化する仕組みとして，ブロックチェーンとスマートコントラクトを活用する．

2.2 IoT 機器のソフトウェア更新手法

現在インターネットに接続する多くの機器が OTA 機能を搭載し，継続してアップデートが提供されている．携帯電話やワイヤレスイヤホン，自動車など様々分野で OTA 機能は提供されているが，製品を開発する企業・団体で OTA 機能の統一的な手法が確立していない．今後 IoT サービスの拡大に応じて，異なる企業・団体が開発した機器の連携が可能であることが必要だと考えられる．インターネット技術の標準を策定する Internet Engineering Task Force（IETF）でも，IoT 関連技術の標準化の議論が行われており，IoT 機器の遠隔管理についても対象になっている．その中でファームウェアアップデートについての標準化を取

り扱うワーキンググループである Software Updates for Internet of things (SUIT) WG[14]では，更新の仕組みの検討が進められている．SUIT では，悪意を持ったファームウェアでアップデートされることを防ぐ必要があると述べているが，ハードウェアリソースの少ない機器や低品質なネットワークに接続しているなど，機器の接続環境によっては既存のセキュリティ対策では十分ではないと考えられる．

Mender[15] は，組込み Linux 向け OTA 管理アーキテクチャであり，機器が実行しているアプリケーション，及び機器全体のシステムの両者を更新可能である．アーキテクチャは Mender クライアントと Mender 管理サーバの 2 つで構成されており，中央集権的な構造を取っている．クライアントが定期的なポーリングによって，更新を確認しダウンロードと適用を実行する Mender は，機器認証，ソフトウェアの完全性，クライアントサーバ間の通信経路の保護を考慮しており，公開鍵暗号方式や，楕円曲線暗号，Transport Layer Security（TLS）などの技術を利用している．

本研究では，IoT 機器として Espressif System 社 ESP32[16]を搭載した機器を想定する．ESP32 は，Wi-Fi と Bluetooth を内蔵した低価格，低消費電力な SoC（System on Chip）で，Arduino ボードや M5Stack などの試作用組込みボードで採用されている．ソフトウェア更新は，Arduino C/C++[17]の Wi-Fi 機能及び，HTTP 通信を利用した OTA 機能[18]で実現可能である．本研究では，このソフトウェア更新機能をベースに，ソフトウェアの改竄を防止（完全性を保証）する仕組みを加えた上で，IoT 機器とブロックチェーンを接続してフレームワークを構築する．詳細は，第 3 章で述べる．

2.3 ブロックチェーンを活用した IoT 機器のソフトウェア更新フレームワーク

長柄ら[5]は，ローカルネットワーク内の IoT 機器を管理するパーソナルコンピュータ（PC）や Wi-Fi アクセスポイント（AP）を機器管理ノードとして，配下の IoT 機器を管理する手法を提案している．PC と AP は，ブロックチェーンと通信し，アップデートの有無やアップデートファイルのダウンロードとその管理を実行する方式で，IoT 機器が直接外部と通信することなくアップデートできる．ブロックチェーンは，OSS[19]を参考に開発し，シミュレーションで実現可能性を評価している．

He ら[4]は，ブロックチェーンを利用することで，IoT 機器のアップデートフレームワークを従来のクライアントサーバモデルから単一障害点の排除及び耐改竄性を有したアーキテクチャにすることができると述べている．この研究では，IoT 機器の想定として Wemos D1 Mini Board と Hyperledger を利用してアップデートの実証を行なっている．このアップデート方式は機器を http サーバとして公開し，アップデートファイルを機器開発者が直接機器に送信する形式である．機器は受け取ったアップデートファイルのハッシュ値をブロックチェーンに問い合わせることで検証を行う．

Dhakal ら[6]は，ブロックチェーンがルータなどのネットワークを構成する機器の設定ファイルをセキュアかつ速やかに配布する手段として利用可能であると述べている．ブロックチェーンの分散台帳としての性質は，従来の中央集権型な管理手法に比べて，単一障害点の排除とスケーラビリティ増大による信頼性と可用性の向上が可能であり，ブロックチェーンに書き込まれた情報は記録が改竄される可能性が極めて困難になる．ネットワークの機器管理者はブロックチェーンを通じて設定ファイルを配布することで間接的に機器を管理する．

Maloney ら[7]は，コージェネレーションプラントの分散制御システム内の機器管理を行うフレームワークにブロックチェーンを導入することを提案している．プラント機器管理者は新しい機器をシステム内に配置する際に行う機器の初期設定や定期的なセキュリティチェック，ファームウェアアップデートを行う必要がある．この研究では，定期的なセキュリティチェックやファームウェアアップデートをブロックチェーンの導入によって自動化することを提案している．ブロックチェーンに Ethereum を採用し，ファームウェアの更新や証明書・公開鍵の管理にブロックチェーンを利用して自動化する手法となっている．市販されているルータと Ethereum で実験を行い，システムの実証を行なっている．

Woei's ら[8]は，大容量のストレージスペースの必要性やファームウェアの集中管理等，それまで発表されていたスキームの欠点を解決する為に，ブロックチェーン技術に基づいた保護メカニズムが提案されている．ブロックチェーンには Ethereum を採用し，分散データストレージを用いることで，ファームウェアの集中管理を無くしている．またファームウェアのアップロード及びダウンロードの際に，独自のアルゴリズムを用いることでファームウェアの整合性を検証している．ファームウェアの整合性や IoT 機器の接続及びシステムのセキュリティ等で，いくつかの先行研究との比較を行い，セキュリティの高さを示している．

上記の既存研究では，ブロックチェーンを導入して，堅牢なソフトウェア更新フレームワークを実現できることを述べている．一方で，機器利用者が，これらのフレームワークを利用する観点で，使用性やセキュリティについて議論されていない．例えば，[4][5]では，ソフトウェア更新フレームワークのステークホルダとして，機器利用者が登場しない．[6][7]では，機器管理者と機器利用者を区別せず，機器のソフトウェア更新と利用を同一人物が実施することを想定している．ソフトウェア更新フレームワークの実用性を議論するには，フレームワークに対する要求，ユースケースを整理した上で，具体的な設計や実装を，誰でも自由に変更できる環境の構築が望ましい．特に，実際に動作可能で，かつ設計や実装を公開している研究は，我々の知

る限りでは存在しない．本論文では，これらの問題を解決するために，IoT 機器向けソフトウェア更新フレームワークの実用を想定した，要件とユースケースを示し，要件とユースケースにもとづいて，提案フレームワークの具体的な設計と実装の事例を示す．将来的には，ソースコードを公開し，オープンソースソフトウェアとして公開すること目指している．

3. 提案フレームワークの設計と実装

3.1 概要

ステークホルダごとに，ソフトウェア更新に関する要求を分析し，ソフトウェア更新フレームワークに対する要件を定義する。次に，要件を満たす提案フレームワークの設計と実装について述べる．

1. ステークホルダ定義と要求分析
2. 要求に基づく機能要件定義
3. 機能要件を満たすフレームワーク設計
4. 設計したフレームワークの実装

なお，ここで示すステークホルダの定義と要求分析の結果は，我々の研究グループで実施した結果である．より実用性を高めるには，今後，実際の機器開発や保守の経験のある技術者と議論を進める必要がある．

3.2 要求分析

3.2.1 ステークホルダ

Web カメラ，スマートロック，家電など，一般に購入できる複数の IoT 機器を，セキュリティに関する専門知識の乏しい一般利用者が，Wi-Fi を利用可能な閉鎖空間（自宅や職場等）管理する状況を想定する．想定するステークホルダを以下に示す．

機器開発者

- IoT 機器及び搭載されるソフトウェアを開発し，脆弱性を修正するソフトウェア更新にも責任をもつ．ソフトウェア配布用リポジトリを管理，運営する．

機器管理者

- IoT 機器の利用者で，IoT 機器を購入後に設定，利用，廃棄に責任をもつ．IoT 機器に応じたソフトウェア更新処理を実施する．

ソフトウェア更新フレームワーク管理者

- ソフトウェア更新フレームワークを管理，運営する

3.2.2 機器開発者の要求

機器開発者にとっては，確実かつセキュアなソフトウェア更新を実現し，負担をできる限り軽減することが望ましい．例えば，大量に出荷した IoT 機器の機器管理者に対して，それぞれに更新状況を通知するのは負担が大きい．更新ソフトウェアが配布され，IoT 機器にダウンロードされるまでに，ソフトウェアの改竄や盗聴によって，機器気両者の安全性やセキュリティがしないよう十分に考慮すべきである．さらに，IoT 機器の種別に依存せず，統一した手法でソフトウェアを配布，更新できることが望ましい．よって，機器開発者の要求を以下のように整理する．

- 大量に出荷した IoT 機器に対して，確実に，効率的かつセキュアにソフトウェアを配布できること
- 多種多様な IoT 機器に，統一的に適用できること

3.2.3 機器管理者の要求

機器管理者にとっては，IoT 機器の安全性やセキュリティを確保しながら，継続して使用できることが望ましい．具体的には，所有する多種多様な機器の利用状況，ソフトウェアのバージョン，更新履歴を確認し，必要に応じて，ソフトウェアを統一的な方法で更新できることが望ましい．機器利用者の要求を以下のように整理する．

- 機器購入後に初期設定さえすれば，ソフトウェアの更新を自動化できること
- 単にソフトウェア更新だけでなく，機器の利用状況やソフトウェア更新履歴を自動的に残すこと

3.3 要求に基づく機能要件定義

3.3.1 機器開発者の機能要件

機器開発者の要求に基づき，提案フレームワークの機能要件を検討した．

機器開発者登録機能

- 概要：機器開発者をフレームワークに登録する．
- アクター：機器開発者
- 事前条件：なし
- 入力：認証情報
- 基本フロー：1.認証情報の入力

IoT 機器登録機能

- 概要：機器開発者が新規に開発した IoT 機器をフレームワークに登録する．認証された機器開発者はフレームワークにアクセスし，機器毎に必要なソフトウェア更新情報を登録する．
- アクター：機器開発者
- 事前条件：機器開発者がフレームワークに登録済であること
- 入力：認証情報，機器情報（機器名，初期ソフトウェアのハッシュ値，URI）
- 基本フロー：1.認証情報を利用して機器開発者を認証，2.機器情報をフレームワークに登録

ソフトウェア更新登録機能

- 概要：機器開発者は，ソフトウェアを更新して配布する際に，更新ソフトウェアの情報を登録する．
- アクター：機器開発者
- 事前条件：提供先となる IoT 機器が機器開発者によって登録されていること
- 入力：認証情報，機器情報(機器名，初期ソフトウェアのハッシュ値，URI)
- 基本フロー：1.認証情報を利用して機器開発者を認

証，2.新規アップデート情報を登録

3.3.2 機器管理者の機能要件

機器管理者の要求に基づき，提案フレームワークの機能要件を検討した機器管理者の要求を満たす機能要件を以下に示す．

機器管理者登録機能

- 概要：機器の管理者（多くの場合は機器の利用者に一致）をフレームワークに登録する．管理者の認証は公開鍵認証方式を想定している．
- アクター：機器管理者
- 事前条件：なし
- 入力：認証情報（機器管理者の公開鍵）
- 基本フロー：1.認証情報を入力，2.機器管理者の認証情報を格納

機器管理登録機能

- 概要：機器管理者が新しい IoT 機器を管理する際にフレームワークに登録する．機器管理登録が完了すると，機器とフレームワークが接続され，ソフトウェア更新が可能になる．
- アクター：機器管理者，IoT 機器
- 事前条件：機器管理者の登録が完了していること
- 入力：機器管理者の認証情報（公開鍵），機器 ID,
- 基本フロー：1.機器管理者が必要な入力情報をフレームワークに登録，2.機器が秘密鍵と管理 ID を利用してフレームワークと通信，3.フレームワークが機器管理者に登録完了通知

ソフトウェア更新履歴閲覧機能

- 概要：機器管理者が IoT 機器のソフトウェア更新履歴を確認する．
- アクター：機器管理者
- 事前条件：機器管理者登録と機器管理登録が完了していること
- 入力：機器管理者の認証情報，機器 ID
- 基本フロー：1.認証情報を利用して管理者を認証，2. 機器 ID で指定された機器のソフトウェア更新履歴を出力（機器 ID が未指定の場合，全ての機器の更新履歴を出力）

3.3.3 IoT 機器の機能要件

機器開発者と機器管理者に関連する機能要件の検討を通じて，提案フレームワークに対応するために IoT 機器がもつべき機能要件を整理した．

ソフトウェア更新問合せ機能

- 概要:IoT 機器がフレームワークに対してソフトウェア更新の有無を問い合わせる．
- アクター：IoT 機器
- 事前条件:IoT 機器がフレームワークに登録済みであること
- 入力：機器の認証情報，機器情報（機器 ID，現在のソフトウェアのバージョン情報）
- 基本フロー：1.機器情報をフレームワークに送信，2. 対象機器の最新のソフトウェアと，機器のソフトウェアのバージョンを比較し，新しいバージョンのソフトウェアの有無を機器に返す

ソフトウェア更新履歴の追加機能

- 概要：ソフトウェアの更新が完了した後に，完了したことを，ソフトウェア更新履歴に追加する．
- アクター：IoT 機器
- 事前条件:IoT 機器がフレームワークに登録済であること
- 入力：機器の認証情報，機器情報（ソフトウェアのバージョン情報，更新した日時)
- 基本フロー：1.認証情報を利用して機器を認証，2. ソフトウェア更新履歴に，現在のバージョン情報と日時を追加

3.4 アップデートフレームワークの設計

3.4.1 全体構造

設計した提案フレームワークの全体構造を図 3.1 に示す．中央部分にあるブロックチェーンで実行される 2 種類のスマートコントラクトが，前節で定義した要件を満たす機能を実現するのが特徴である．ブロックチェーンの環境構築が容易で，スマートコントラクトを開発できることから Ethereum を採用する．ソフトウェア配布管理スマートコントラクトは，ソフトウェアの更新情報を提供，ソフトウェア更新に必要となる実行ファイル（バイナリ）は，機器開発者が管理するリポジトリに登録される．機器管理者，機器開発者と IoT 機器は，スマートコントラクトとの通信を通して，フレームワークが提供する機能を利用する．機器開発者と機器管理者がブロックチェーンとやり取りするには，いくつかの方法があるが，より容易に使えるよう専用のアプリケーション（Dapps）を使用する。

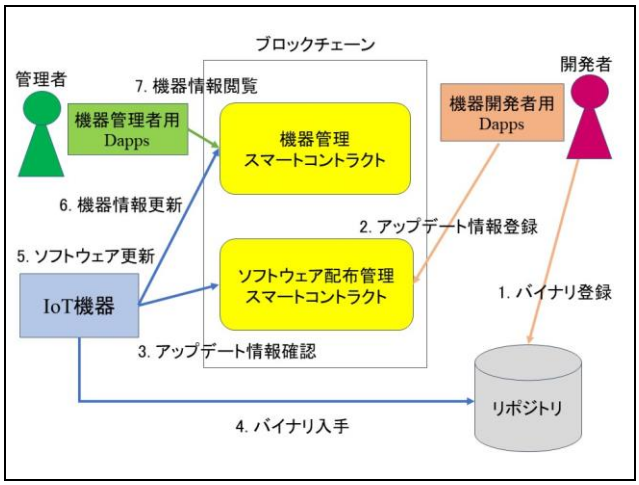

図 3.1 提案フレームワークの全体構造

3.4.2 ソフトウェア配布管理スマートコントラクト

ソフトウェア配布管理スマートコントラクト（ソフトウェア管理 SC）は，IoT 機器に対して新しいソフトウェア情報を提供するコントラクトである．機器ごとに，機器管理者がブロックチェーンに登録（デプロイ）する．機器管理者は，新しいソフトウェアを配布する場合，ウェブサーバ（例えば，GitHub や独自 Web サーバ）にソフトウェアのバイナリを配置し，その URI とハッシュ値をソフトウェア管理 SC に登録する（ソフトウェア更新登録機能）．IoT 機器は，ソフトウェア管理 SC に登録された情報を閲覧することができ，新しいソフトウェアの更新の有無を問合せる（ソフトウェア更新問合せ機能）ことができる．

3.4.3 機器管理スマートコントラクト

機器管理スマートコントラクト（機器管理 SC）は， IoT 機器のバージョン情報及びアップデート履歴を管理するコントラクトである．機器ごとに，機器管理者がブロックチェーンにデプロイする．IoT 機器がソフトウェア管理 SC にソフトウェア更新を問合せ，ソフトウェアを更新したら，そのバージョン情報と更新日時を機器管理 SC に登録する（ソフトウェア更新履歴の追加機能）。機器管理者は，管理下にある機器について，ソフトウェア更新履歴を確認する際には，機器管理 SC に問い合わると確認できる（ソフトウェア更新履歴閲覧機能）．

3.4.4 機器開発者と機器管理者用インタフェース

機器開発者と機器管理者がブロックチェーンとやり取りするには，いくつかの方法がある．Ethereum では，スマートコントラクトの標準開発環境として，Remix が用いられるが，ブロックチェーンや Ethereum に詳しくない機器開発者や機器利用者には使いづらい．そこで，より簡単に提案フレームワークを使えるよう専用アプリケーションを開発している．詳細は 3.6 節で述べる．

3.5 フレームワークの利用シナリオ

提案フレームワークの動作を説明するために，以下の４つの利用シナリオにもとづくシーケンス図を示す．

- 機器開発者による新しい IoT 機器の登録
- 機器管理者による IoT 機器の管理登録
- IoT 機器によるアップデート非適用と履歴更新
- IoT 機器によるアップデート適用と履歴更新

3.5.1 機器開発者による新しい IoT 機器の登録

シナリオ１は，機器開発者が提案フレームワークを通じて新しい IoT 機器にアップデートを提供する際の動作である．はじめに，新しい機器に対してアップデート情報を提供するスマートコントラクトを機器開発者が作成する．初期情報として，IoT 機器名，初期バージョン情報，初期ファームウェアの URI，及びハッシュ値を登録した状態でブロックチェーン上に展開する．このスマートコントラクトへアクセスする為に，必要な認証情報はあらかじめ機器開発者が機器に組み込む必要がある．

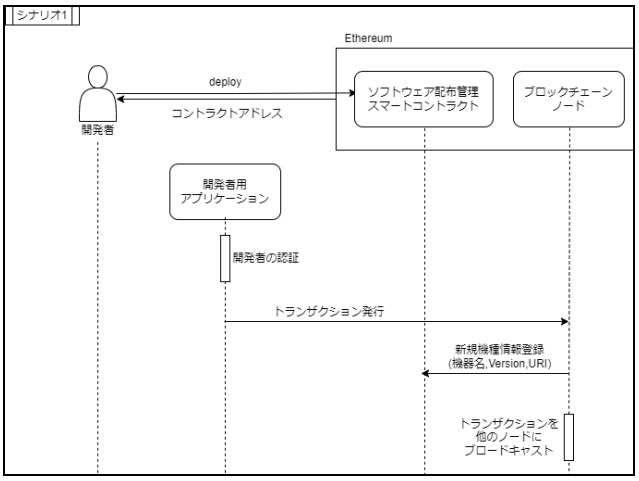

図 3.2 機器開発者による新しい IoT 機器の登録

3.5.2 機器管理者による IoT 機器の管理登録

シナリオ 2 は，機器管理者が提案フレームワークを通じて管理可能な IoT 機器を登録する場合の動作である．IoT 機器ごとに必要な機器管理スマートコントラクトは機器管理者があらかじめ作成しておく．作成時に初期バージョン情報と，初期化されたタイムスタンプを登録しておく．機器管理者が機器管理スマートコントラクトをデプロイする際に，機器管理スマートコントラクトに機器管理者情報を登録する．機器はコントラクトと最初の通信を行う際に機器管理者情報を登録する．これ以降機器管理者は機器管理スマートコントラクトに対してアクセスが可能になる．

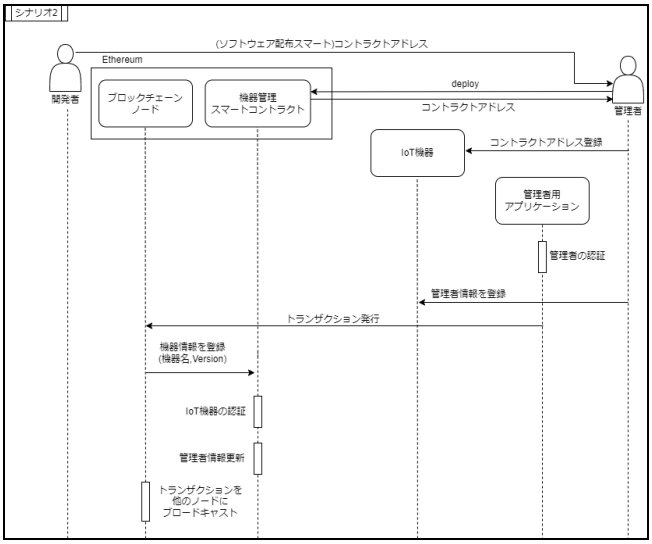

図 3.3 機器管理者による IoT 機器の管理登録

3.5.3 IoT 機器によるアップデート非適用と履歴更新

シナリオ 3 は平常時の IoT 機器のアップデートの確認とその管理を行う場合のものである．IoT 機器は定期的にアップデート提供スマートコントラクトに対して，アップデートの有無を問い合わせる．新規のアップデートが存在しない場合は，スマートコントラクトは現在のバージョン情報のみを返し，それを受け取った IoT 機器もアップデート処理を実行しない．アップデートの有無にかかわらず，IoT

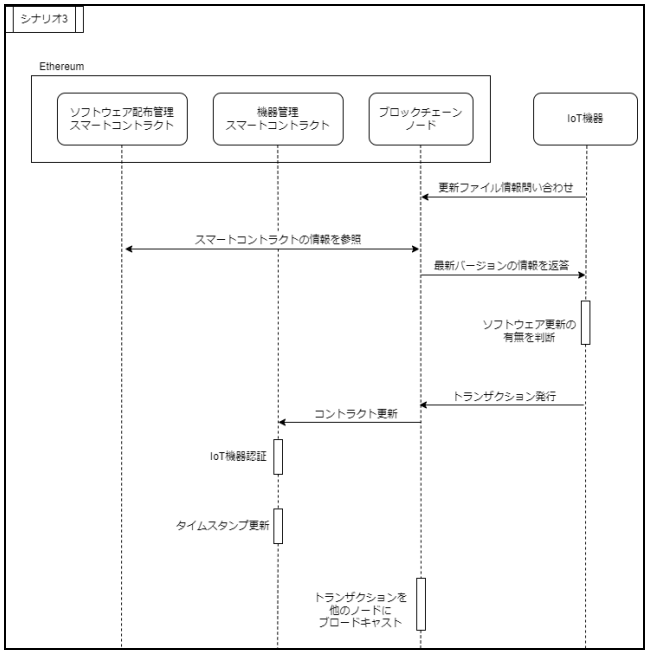

図 3.4 IoT 機器によるアップデート非適用と履歴更新

機器は機器管理スマートコントラクトに対して，問い合わせタイミングのタイムスタンプを更新する. 機器管理者は，機器のバージョン情報やアップデート履歴を機器管理スマートコントラクトに対して問い合わせることで確認することが可能である.

3.5.4 IoT 機器によるアップデート適用と履歴更新

このシナリオ 4 は，IoT 機器に対して新しいアップデートがある場合に行われるものである．機器開発者は，新しいアップデート情報をアップデート提供スマートコントラクトに，アップデートバイナリを，自身が管理するリポジトリにアップロードする．IoT 機器は平常時のシナリオの時と同様，定期的にアップデートの有無をコントラクトに問い合わせる．アップデートがある場合，機器に対して新しいバージョン情報とアップデートバイナリの URI とハッシュ値を応答として返す. 機器は入手した URI からリポジトリにアクセスし，アップデートバイナリをダウンロードする．ダウンロードしたアップデートバイナリによって，アップデートを実行する．アップデートが完了次第，新しいバージョン情報とタイムスタンプを機器管理スマートコントラクトに登録する.

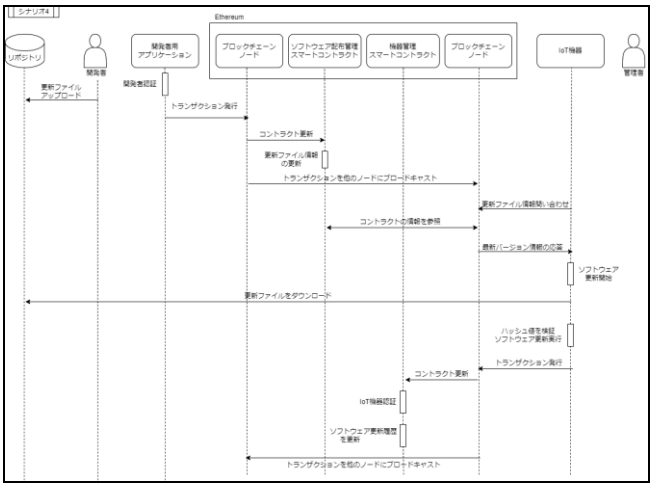

図 3.5　IoT 機器によるアップデート適用と履歴更新

3.6 実装

3.6.1 ブロックチェーンとスマートコントラクト

本フレームワークでは，ブロックチェーンとして Ethereum[20]を，スマートコントラクトの記述では Solidity 言語[21]を用いた．Ethereum ノードとして PC を 1 台使用し，Geth v.1.10.17-stable でテスト用 Ethereum ネットワークを構築した．Ethereum にデプロイされたスマートコントラクトによってトランザクションを発行したり，スマートコントラクト内の情報を参照したりするには，Ethereum を構成する PC に対して HTTP 通信で JSON 形式のメッセージを送信する JSON-RPC[22]で実装した.

ソフトウェア管理 SC と機器管理 SC は，それぞれ 100 行程度の Solidity プログラムである．各機器には，機器 ID として，Ethereum アドレスを登録することで，あらかじめ決められた機器のみ，スマートコントラクトに対してアクセスが可能となる.

3.6.2 機器開発者と機器管理者用アプリケーション

機器開発者用アプリケーションは，機器開発者が実施する IoT 機器登録とソフトウェア更新登録機能を実現するためのインターフェースである．機器開発者用アプリケーションは，機器管理者が実施する管理機器登録機能と，ソフトウェア更新履歴閲覧機能を実現するためのインターフェースである．どちらも，JavaScript ライブラリである React と，Ethereum Wallet である MetaMask を用い，Web ブラウザでブロックチェーン上のスマートコントラクトを操作できるアプリケーションとして開発している.

3.6.3 IoT 機器

IoT 機器として, 無線 LAN に接続可能なチップ ESP32[23]を搭載した機器を想定している．本研究では，ESP32-D0WD-V3 を搭載した M5Stack Core2 IoT を用いた. M5Stack Core2 IoT には, を搭載した IoT 機器を対象とする. ESP32-D0WD-V3 は，ESP32 ベースで, 動作周波数 240MHz のデュアルコア，520KB の SRAM，及び 4MB のフラッシュメモリを搭載している．ESP32 は，Arduino C/C++[17]で

ソフトウェアを開発でき，Wi-Fi 及び，HTTP 通信を利用した OTA 機能[18]をサポートしている．本研究では，このESP32 の標準的な OTA 機能をベースに，提案フレームワークに対応する OTA 機能を実装した．

ESP32 で OTA を利用する際は，フラッシュメモリのパーティション設定が必要で，最低でも 2 つの OTA app slot と OTA Data Partition の領域が必要である．OTA を実行する際は，現在起動しているものとは異なる OTA app slot にバイナリを書き込み，次の起動先を OTA Data Partition の領域に書き込む．OTA によって新たに書き込まれたソフトウェアが正しく動作しなかった場合にロールバックする仕組みも搭載している．ロールバックをする為に，新たに書き込むソフトウェアサイズに制限が必要であり，ソースコードのサイズは 1.31MB までにする必要がある．

ESP32 における標準的なソフトウェア更新処理の流れは以下の通り[24]である．

1. ESP32 が更新ファイルの URI を指定して HTTP サーバと通信を開始
2. HTTP サーバから更新ファイルをダウンロード
3. ダウンロードしたバイナリを OTA 用領域である OTA app slot に書き込む
4. 次の起動先として新しく書き込んだバイナリを指定して再起動

ESP32 の標準的なソフトウェア構成図を図 3.6 に示す．

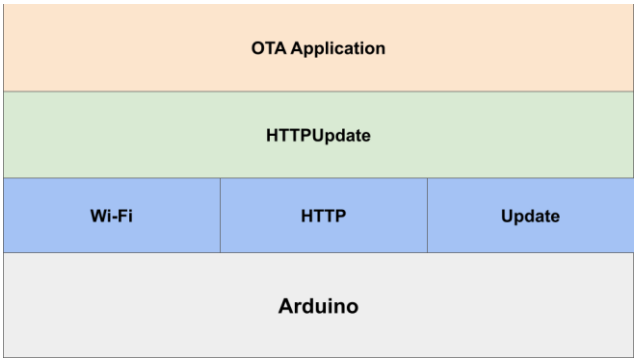

図 3.6　ESP32 の標準ソフトウェア構成

一番下に位置する Arduino ライブラリは，アプリケーションの実行に必要な機能を提供し，標準入出力や各種タイマや LED 制御を担当する．上にある 3 つのライブラリはそれぞれ Wi-Fi，HTTP，Update に必要な機能を提供する．特に Update ライブラリは OTA の重要な機能を実現する．入力としてストリームを受け取り，更新するバイナリのフラッシュへの書き込みと，受け取ったバイナリの改竄を検出するためにハッシュ値を計算する処理を実行する．本研究では，SHA256 で更新バイナリのハッシュを計算する処理を追加実装した．

HTTPUpdate ライブラリは，HTTP サーバを利用してソフトウェアを更新する ESP32 のライブラリである．更新バイナリの URI を受け取り，更新バイナリをダウンロードしたのちに，Update ライブラリを呼出す．これらライブラリを利用する OTA アプリケーションを実行することで，HTTP サーバから新しいソフトウェアをダウンロードし，再起動をすることで OTA が完了する．ESP32 の OTA ライブラリでは，自身を HTTP サーバとして公開し，更新バイナリをアップロードしたのちに更新する方式と，外部に存在する HTTP サーバからダウンロードして更新する方式の 2 種類が存在する．提案フレームワークと接続するために，後者の方式を採用した．

提案フレームワークに接続するために開発した OTA アプリケーションは，以下の 6 ステップで動作する．

1. ESP32 がソフトウェア管理 SC にソフトウェア更新の有無を定期的に問い合わせる
2. ソフトウェア更新がある場合は，ソフトウェア管理 SC で指定された URI に接続し，更新ファイルをダウンロード
3. ソフトウェア管理 SC から受け取ったハッシュ値と，ダウンロードしたバイナリから計算したハッシュ値が一致することを確認
4. ダウンロードしたバイナリを OTA 用領域である OTA app slot に書き込む
5. 機器管理 SC にソフトウェア更新履歴を追加
6. 次の起動先として新しく書き込んだバイナリを指定して再起動

実装した ESP32 のソフトウェア構成図を図 3.7 に示す．

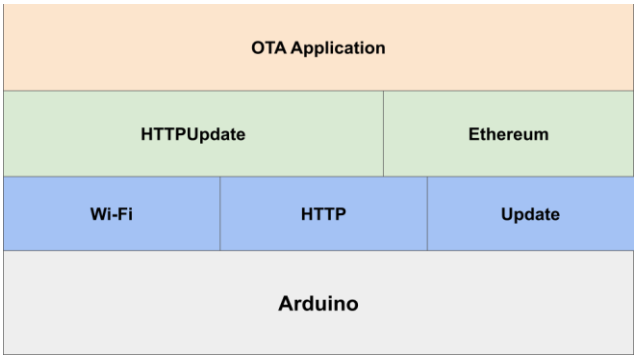

図 3.7　実装したソフトウェア構成

Ethereum ライブラリは Ethereum との通信に必要なライブラリであり，Ethereum に対して送信する JSON メッセージの生成，Ethereum に対して送信，受信したデータの変換などを含む自作のライブラリである．HTTPUpdate ライブラリには，ソフトウェア管理 SC に対するソフトウェア更新問合せ機能と，機器管理 SC に対するソフトウェア更新履歴追加機能を実装した．OTA アプリケーションは，本来は IoT 機器が実現すべき機能が実装されるものであるが，本研究では，提案フレームワークの動作に必要な最低限の機能として，先に示した 6 ステップの処理のみを実装した．

4. 提案手法の評価

4.1 定性的評価

評価では，フレームワーク構造，ステークホルダの負担，セキュリティの観点で，既存研究と定性的に比較し，提案フレームワークによって，ステークホルダの負担を減らせる可能性について評価する．

比較対象は，2 章で述べたブロックチェーンを利用した更新フレームワークと Amazon Web Service IoT（AWS IoT）[25]である．各表における記号の意味は，以下の通りである．

- ○：当該手法のみで実現可能
- △：当該手法と他技術を組み合わせることで実現できる可能性がある
- ×：当該手法では実現が難しい

4.1.1 フレームワークのアーキテクチャ

表 4.1 にアーキテクチャの観点で比較結果を示す．

表 4.1 アーキテクチャの比較

	提案手法	[5]	[4]	[6]	[7]	[8]	[25]
ソフトウェア更新履歴の管理	○	×	△	△	△	△	○
履歴の機器管理者への提供	○	△	×	△	△	△	△
ソフトウェア更新の効率性	○	○	×	○	△	○	○
多種多様な機器への適用可能性	○	○	○	×	○	○	△

機器のソフトウェア更新履歴について，提案手法では，更新確認日付と更新実行日付，機器のバージョン情報を，ブロックチェーン上のスマートコントラクトで管理する．[5]では，PC/AP 経由で IoT 機器の更新を実行するので，PC/AP で配下の機器の情報管理が可能である．加えて，ブロックチェーン上でも更新の実行状況を管理する．PC/AP では，機器開発者向けのログ管理を実行していると考えられる．PC/AP で履歴を管理すると，その履歴データの信頼性は PC/AP に依存するので，故障によるデータの喪失や，機器交換に伴う履歴の移し替えに対応する必要があるのが欠点であると考えられる．[4]や[6]は，Hyperledger を利用したフレームワークであり，ブロックチェーン上で更新履歴を管理できる．しかし，パーミッション型ブロックチェーンであるため，ステークホルダが直接ブロックチェーンにアクセスすることができない．すべてのステークホルダの真正性を認証してアクセス権限を管理することも可能と思われるが，その仕組みについては言及されていない．[7]と[8]では，Ethereum のトランザクションとしてログを残す方法を採用しているが，トランザクションの中身を確認するにはブロックチェーン内のブロックを検索する必要がある．ブロックチェーンの検索性は低いことが知られており，この方法は効率的ではない．AWS IoT[25]を利用する場合は，機器開発者が発行するソフトウェア更新の指令（AWS ではジョブと呼ばれている）の履歴は全て記録されており，機器開発者はいつでも参照可能である．しかし，そのソフトウェア更新履歴を機器管理者に提供する仕組みが存在しない．機器管理者にそのような情報を提供するには，別途，機器利用者用システムを導入し，AWS 上の履歴を提供する仕組みが必要となる．

ソフトウェア更新の効率性と適用可能性について，提案手法では，スマートコントラクトを通じてソフトウェア更新情報を提供し，それを受けた機器が更新処理を開始する流れである．[5]では，PC/AP 経由で提案手法と同様の流れであり，効率性や適用可能性に関しては差がないと考えられる．[4]は，機器を HTTP サーバとして公開し，ベンダが直接，更新ファイルを送信して更新を実行する方式である．機器を常時サーバとして公開するのは，DoS 攻撃などに晒される危険性が高くなるので，更新を行うタイミング時のみ公開することが望ましいが，その実現は容易ではない．[6]では，ブロックチェーンを経由して設定ファイルを配布する為，効率性は良いと考えられるが，ブロックチェーンに格納できるデータサイズ制約の観点から，ファームウェアなどの更新ファイルのサイズが大きい機器の OTA には適用が難しいと考えられる．[7]は，トランザクションに更新情報を記載するので，更新が必要な機器が持っているアドレス宛のトランザクションを検索する必要がある．スマートコントラクトを経由せず，トランザクションを直接検索するので，提案手法に比べて，その効率性は低い[17]．[8]は，提案手法と同様にブロックチェーン上には更新ファイルのアドレスのみを保存しており，効率性や適応可能性に関して差がないと考えられる．AWS IoT[25]は，機器開発者が更新のジョブを発行すると，IoT 機器はそのジョブから更新が必要であると判断し，更新ファイルを入手する．ジョブは機器単位だけでなく，機器のグループ単位で発行できるので，効率的に更新ファイルを配布可能である．しかし，対象とする IoT 機器は現在限られており，多種多様な機器において実現するには，AWS IoT の利用を前提とした機器開発が必要となる．

4.1.2 セキュリティの比較

表 4.2 セキュリティの比較

	提案手法	[5]	[4]	[6]	[7]	[8]	[25]
通信の機密性	△	○	○	○	△	○	○
通信の完全性	○	○	○	○	○	○	○
開発者の真正性	○	○	○	○	×	○	○

フレームワークが満たすべきセキュリティの性質は，機器開発者から IoT 機器までのエンドツーエンドで更新ファイルの機密性と完全性が保証されていることである．これに関して，提案手法で実現する為には，機器とリポジトリ間で更新ファイルが保護されることが重要である．通信の機密性については，提案手法ではブロックチェーンにパブリックチェーンを利用しているので，ブロックチェーンの

すべての参加者には，トランザクションが公開される．提案手法の現仕様では，機器管理 SC へのトランザクションがすべてのブロックチェーンノードに対してブロードキャストされるので，ソフトウェア更新問合せのトランザクションと，ソフトウェア更新完了のトランザクションが，ブロックチェーン参加者に公開される．更新ファイルの機密性について保護する場合は，機器とリポジトリ間の通信を暗号化する処理を追加で実装する必要がある．[4], [5], [6], [25]では，SSL/TLS を導入可能な場合はすべて導入すると述べているので，機密性の確保が可能であると考えられる．

（更新ファイルを含む）通信の完全性について，提案手法では，更新ファイルのダイジェストであるハッシュ値がブロックチェーン上のスマートコントラクトに格納されている．そのハッシュ値を利用することで更新ファイルの改竄を検出できる．この機能は，他の先行研究でも導入された方式であり，ブロックチェーンを利用したアップデート手法の標準的な方法になると考えられる．[8]では，更新ファイルのアップロードとダウンロードの際に，独自アルゴリズムによって整合性を検証することで，通信の完全性を保証している．

機器開発者の真正性が保証されていないと，不正なソフトウェアで機器が更新されてしまう可能性がある．提案手法では，ソフトウェア管理 SC を通じて，更新を通知する．このスマートコントラクトを更新可能なのは，ブロックチェーン上にあらかじめ登録された機器開発者のみである．このスマートコントラクト経由でアップデート情報が通知されるので，不正なアップデート情報がブロックチェーン経由で配布されることはないと考えられる．機器管理者登録においては，当然ながら真正性の保証に配慮した手続きが求められるが，本研究ではまだ十分に検討できておらず，今後の課題である．

4.2 使用性評価

4.2.1 ステークホルダの負担比較

機器開発者と機器管理者の観点から利用時の負担を比較した．フレームワークの運営では，パブリック型ブロックチェーンを採用すれば，ステークホルダがフレームワークの運営に関与する必要はない．パーミッション型ブロックチェーンを利用する際には，ステークホルダ間で合意された運営が必要となる．ここでは，フレームワーク運営は，機器開発者の負担に関わると判断し，比較項目に追加した．列挙した．表における記号の意味は，以下の通りである．

- ○：当該手法のみでステークホルダに負担がない，もしくは小さい
- △：当該手法で実現可能だがステークホルダの負担が大きいと思われる
- ×：提案手法では実現が不可能

表 4.3 機器開発者負担の比較

	提案手法	[5]	[4]	[6]	[7]	[8]	[25]
バイナリ配布の容易さ	○	○	×	○	△	○	○
フレームワーク運営負担	○	△	△	△	○	○	○

機器開発者の負担としては，全手法に共通して，更新ソフトウェアの開発と，リポジトリへのアップロードが必要である．提案手法では，ソフトウェア更新 SC に対して，更新ソフトウェアの情報を更新し，リポジトリに更新バイナリをアップロードする作業が必要である．[5]でも，同様の作業が必要である．[4]では，機器開発者が更新対象の機器に対して，更新バイナリを直接送信する方式を採用している．この手法は，対象となる機器数の増加に伴って，機器開発者の負担も増大する可能性がある．[6]では，機器の設定ファイルというサイズが非常に小さいファイルを配布するので，直接，ブロックチェーン上に設定ファイルを格納している．ブロックチェーン上に格納されたデータは，ブロードキャストされることで全てのノードに共有される．IoT 機器の更新バイナリのサイズは，アプリケーションの規模や，機器のフラッシュメモリのサイズに依存するが，数メガバイト以上のサイズであると想定すると，バイナリ自体をブロックチェーンに格納する方法は，ネットワーク帯域の消費，トランザクション発行における手数料などの観点からも望ましくない．[7]では，ソフトウェア更新をスマートコントラクト経由ではなく，ブロックチェーンに発行したトランザクションを IoT 機器自身が監視し，自分宛のトランザクションをチェーンで発見した際に，格納された更新情報を入手する方法を採用している．機器の Ethereum アドレスを機器毎に分けると，機器数だけトランザクションを発行する必要があり，機器開発者の負担は大きく，処理効率も低い．[8]では，更新ファイルを分散データベースにアップロードする際に，更新ファイルの安全性を検証する．機器開発者は，安全性の基準を満たした更新ファイルの開発が求められるが，それ以外は負担が小さい．AWS IoT[25]では，ジョブと呼ばれる機器に対する指令を AWS 上で発行し，署名したアップデートファイルをクラウドストレージにアップロードする．機器はジョブを確認したのちに，ストレージに更新ファイルを問い合わせ，ダウンロードして更新処理を実行する．同じ種類の機器に対しては，まとめてジョブを発行できるので，小さな負担で更新バイナリを配布できる．

フレームワーク運営の負担に関しては，提案手法と[7]では，パブリックチェーンである Ethereum を利用するので，ブロックチェーンを運営するコストは低い．[4-6]は，パーミッション型チェーンを利用するので，前述の通り，運営上の負担がある．AWS IoT[25]を利用する場合は，Amazon 社がフレームワークを運営するため，負担は小さいが，長期に渡って利用料金を払い続けることになり，費用面での負担は大きな課題となり得る．

表 4.4　機器管理者負担の比較

	提案手法	[5]	[4]	[6]	[7]	[8]	[25]
更新の容易さ	○	○	○	△	△	○	○
バージョン管理の容易さ	○	△	△	○	×	△	△

ソフトウェア更新は, 提案手法, [4], [5], [8], AWS IoT[25]で，特に機器管理者の負担は小さい．機器を購入した際の設定作業を済ませた後は，機器管理者の負担がなく更新可能であると考えられる．[6]と[7]では，機器利用者が機器管理者の役割も担い，ソフトウェアの開発と配布，更新を実施する想定で，負担が大きい．AWS IoT[25]を利用する場合も, 機器管理者は, AWS に関する情報や知識がなくても，機器が自動的に更新の確認から入手，適用まで行ってくれるので機器管理者の負担は存在しない．

バージョン情報や更新履歴の管理については，提案手法ではスマートコントラクトを用いて，管理者自身が機器に対して特別な処理を行うことなく管理･閲覧が可能である．それに対して，[5]では PC/APP の管理が必要になる．[4]では，Hyperledger Fabric のトランザクションログを外部の DB に転送する可視化ツール[26]を利用しているが, 機器管理者が閲覧するには，DB を利用した別システムを新たに構築する必要がある．[6]では，ブロックチェーン上で管理との記載がある為，提案手法と同様スマートコントラクトを用いた履歴管理を行うことが可能である．[7]では，履歴を得るためにはトランザクションを直接検索する必要があるので, 機器管理者に時間的負担がある. [8]と AWS IoT[25]は，現時点においては，機器利用者向けに機器のバージョン管理情報などを提供していない．

4.3 定量的評価

4.3.1 評価の概要

ソフトウェア更新機能の実行時間と，IoT 機器におけるメモリ消費量の観点で，定量的に評価し，提案フレームワークの実用性を評価する．具体的には，IoT 機器の RAM とフラッシュメモリの 消費量, ローカルネットワーク環境における平均ソフトウェア更新時間を測定した．それら項目を Arduino 環境で提供されている標準 OTA 手法と AWS IoT（Amazon Web Services で利用可能な Amazon FreeRTOS[27]の OTA 方式）を対象に比較した．

4.3.2 RAM 消費量及びフラッシュ消費量の評価方法

RAM 消費量とフラッシュメモリ消費量は，提案手法と Arduino 標準 OTA 手法では，Arduino IDE でビルドした際に表示される出力ログで確認した．Amazon FreeRTOS の場合は，RAM 消費量に関しては仕様書[28]から，フラッシュメモリ消費量に関してはバイナリサイズを直接計測した．

測定結果を表 4.5 に示す．

表 4.5　RAM とフラッシュメモリの消費量

	提案手法	標準 OTA	AWS IoT
RAM 消費量	39KB	38KB	65KB
フラッシュメモリ消費量	902KB	719KB	1.0MB

ESP32 標準 OTA 手法と提案手法を比較すると，RAM 消費量が 3%，フラッシュメモリが 24%程度の増加となっている．提案手法では，ライブラリの追加と OTA アプリケーション処理を追加しているので，これらがフラッシュメモリの増加に繋がっている. 提案手法と Amazon FreeRTOS と比較しても RAM 消費量が 40%，フラッシュメモリ消費量が 13%少なくなっており，Amazon FreeRTOS を利用するよりも機器のリソース消費量が低いことがわかる．以上の結果から，提案手法による RAM とフラッシュメモリ消費量の増加は非常に少ないので実用面で問題ないと考える．

4.3.3 平均ソフトウェア更新時間の評価方法

ESP32 標準 OTA と提案フレームワークでの OTA の流れは 3.6.3 節で詳細に述べた．Amazon FreeRTOS と AWS IoT を利用する場合の OTA の流れは，以下の通りである．

1. ESP32 が MQTT の Subscriber として AWS 上の OTA ジョブが発行されるまで待機
2. 機器管理者が AWS で新しいジョブが発行
3. ESP32 は更新情報を MQTT で受け取り，更新ファイルを AWS クラウドストレージからダウンロード
4. ダウンロードした更新ファイルの真正性と完全性を署名で確認
5. AWS に完了通知を送信

更新処理の測定では，更新開始と終了までの時間を各手法で表 4.6 のように定義し, 測定 20 回の平均値を算出した．

表 4.6　平均ソフトウェア更新時間の測定時間の定義

	更新開始	更新終了
提案手法	更新確認メッセージ送信	ソフトウェア更新履歴の更新
標準 OTA	HTTP サーバとの通信開始	更新終了後のメッセージ送信
Amazon Free RTOS	クラウド上で更新要求発行	クラウド上のソフトウェア更新履歴更新

測定環境は，ESP32 標準 OTA については，ローカルネットワーク環境の PC（MacOS Mojave 10.14.6，プロセッサ：2.3 GHz Intel Core i5，メモリ：8GB 2133Mhz DDR3）に，リポジトリとなる HTTP サーバとして Python2.7 Simple http server を起動し，更新終了時に HTTP サーバに対して完了通知を出す処理を追加した．提案手法では，標準 OTA と同じネットワーク環境に PC（Ubuntu20.02，プロセッサ：i5-8365U，メモリ：4GB）を追加し，Ethereum とそのクライアントアプリケーション Geth 1.10.3 を起動し，スマートコントラクトを事前に登録した．AWS IoT は，リポジトリとして Amazon Simple Storage Service を利用し，サーバの位置としてアジアパシフィックを選択した．

平均ソフトウェア更新時間は，標準 OTA で 17.9 [s]，提案手法は 22.4[s]，AWS IoT は 35.5[s]であった．標準 OTA に比べて，提案手法のソフトウェア更新時間が伸びた理由は，ブロックチェーン上でのソフトウェア更新履歴の更新処理（トランザクション）において，ブロックチェーンのマイニングが発生したことが原因だと考えられる．なお，本研究では，実験用の Ethereum を利用しているが，実際のパブリック環境の Ethereum では，1 秒間に 15 個のトランザクション処理速度[29]であり，実際の Ethereum を使用した場合のソフトウェア更新時間はさらに長くなる．AWS IoT では，インターネットを介して機器と AWS が通信しているので，その通信時間が更新時間の差に現れている．今回の実験では，ESP32 標準 OTA に比べて更新時間は長くなったものの，IoT 機器のソフトウェア更新頻度を 1 日に 1 回程度と想定すれば，実用でも問題ないと考える．現時点では機器 1 台のソフトウェア更新時間のみを比較しており，機器数を増加させた場合のスケーラビリティについても評価する必要があるが，これらは今後の課題である．

5. おわりに

本論文では，ブロックチェーンとスマートコントラクトによる自動化を特徴とする，IoT 機器向けソフトウェア更新フレームワークを提案した．機器管理者や機器利用者などのステークホルダと，そのユースケースを分析し，提案フレームワークの要件を整理した．その結果にもとづいて，フレームワークの設計及び実装について詳細に述べた．評価では，既存の関連研究との定性的及び定量的に比較し，提案フレームワークによって，ステークホルダの負担を減らせる可能性と，実用性について評価した．提案手法について，先行研究であるブロックチェーンを利用した他の手法よりも効率的なソフトウェア更新と，ステークホルダの負担を大幅に減らせる可能性を示した．機器の実装においては，メモリ消費量も既存手法と大きな差がなく，多種多様な機器に適用可能であることを確認した．

今後の課題としては，より実用性を高めるに，機器開発者，機器利用者，IoT 機器登録の手順を詳細に設計し，誰でも容易に使えるアプリケーションを開発する必要がある．さらに，実際の機器開発や保守の経験のある技術者や，より多くの研究者とも議論を進めながら，要件や機能を見直すために，本研究の成果をオープンソースソフトウェアとして公開する予定である．

参考文献

[1] 総務省．令和元年版情報通信白書，2019.

[2] 田辺瑠偉,IoT におけるサイバー攻撃の最新動向～IoT マルウェアの多様化～,情報処理学会研究報告コンピュータセキュリティ(CSEC),2021.

[3] Aymen Boudguiga, Nabil Bouzerna, Louis Granboulan, Alexis Olivereau, Flavien Quesnel, Anthony Roger, and Renaud Sirdey. Towards better availability and accountability for iot updates by means of a blockchain. In 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 50–58, 2017.

[4] Xinchi He, Sarra Alqahtani, Rose Gamble, and Mauricio Papa. Securing over-the-air iot firmware updates using blockchain. In Proceedings of the International Conference on Omni-Layer Intelligent Systems, pp. 164–171, 2019.

[5] 長柄啓悟，松原豊，高田広章ほか．ブロックチェーン技術を用いた IoT 機器向けセキュアアップデートフレームワーク．組込みシステムシンポジウム 2018 論文集, Vol.2018, pp. 36–39, 2018.

[6] Samip Dhakal, Fehmi Jaafar, and Pavol Zavarsky. Private blockchain network for iot device firmware integrity verification and update. In 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), pp. 164–170, 2019.

[7] Matthew Maloney, Gregory Falco, and Michael Siegel. Cyber-physical system security automation through blockchain remediation and execution (sabre). 2020.

[8] Woei-Jiunn Tsaur,et al.A Highly Secure IoT Firmware Update Mechanism Using Blockchain, Sensors 2022.

[9] 池内弘樹．許可制ブロックチェーンと既存分散データベースとの性能・実装・運用比較, pp.8-53,2017.

[10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[11] Christian Cachin, et al. Architecture of the hyperledger blockchain fabric. In Workshop on distributed cryptocurrencies and consensus ledgers, Vol. 310, 2016.

[12] Nick Szabo. Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16), Vol. 18, No. 2, 1996.

[13] TiagoMFernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. IEEE Access, Vol. 6, pp. 32979–33001, 2018.

[14] Software updates for internet of things. https://datatracker.ietf.org/group/suit/about/, 2022/1/7 Accessed.

[15] Mender. https://mender.io/, 2022/1/7 Accessed.

[16] David Michel. Conception of a blockchain client for secure transmission of production data. Master's thesis, 2018.

[17] Store Arduino Arduino. Arduino. Arduino LLC, 2015.

[18] espressif. Esp-idf programming guide. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html, 2022/1/10 Accessed.

[19] Github-dvf/Blockchain. A simple blockchain in python. https://github.com/dvf/blockchain, 2021/1/5 Accessed.

[20] Gavin Wood, et al. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, Vol. 151, No. 2014, pp. 1–32, 2014.

[21] Chris Dannen. Introducing Ethereum and solidity, Vol. 1. Springer, 2017.

[22] Matt Morley. Json-rpc 2.0 specification, 2010.

[23] espressif. Esp32 datasheet. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, 2021/1/7 Accessed.

[24] espressif. Esp32 arduino. https://github.com/espressif/arduino-esp32, 2021/1/20 Accessed.

[25] Amazon Web Services IoT. https://aws.amazon.com/iot, 2021/1/10 Accessed.

[26] 森島信,松谷宏紀. GPU を用いたブロックチェーンのフルノードにおける取引検索の高速化．電子情報通信学会論文誌 D,Vol. 102, No. 5, pp. 378 389, 2019.

[27] FreeRTOS. https://www.freertos.org, 2022/1/10 Accessed.

[28] AmazonWeb Services IoT. https://docs.amazonaws.cn/en_us/freertos/latest/userguide/ota-http-freertos.html, 2022/1/10 Accessed.

[29] Shaik V Akram, Praveen K Malik, Rajesh Singh, Gehlot Anita, and Sudeep Tanwar.Adoption of blockchain technology in various

realms: Opportunities and challenges.Security and Privacy, p. e109, 2020.