アップデート中断時の更新プロセスの記録による サーバーからのファイル再受信時の重複排除

宮本 港斗[1]　大沢 恭平[2]　串田 高幸[1]

概要：IoT デバイスは継続的な運用維持のためにバグの修正や, 機能の追加を目的としたアップデートが必要不可欠である. しかし, アップデート中に, 例えばネットワーク障害によりデバイスとのネットワーク接続が切断されると更新プロセスが中断され, 一部のプログラムのみ更新が適用された不完全な状態に陥る. この状況において, 更新プロセスを最初から実行する方法では, 更新に要する時間が延長され, ネットワーク帯域やエネルギーの浪費を招く. 本稿では, サーバーと IoT デバイス間のネットワーク接続が切断されることにより更新が中断された際, デバイスの更新プロセスの記録を行う事による中断された箇所から更新を再開する手法を提案した. サーバー側では IoT デバイスごとの送信したファイル名を記録, IoT デバイス側では更新プロセス中で実行している関数名や更新中のアップデートファイル名をテキストフォーマット形式のファイルに記録する. この記録された関数名と更新ファイル名をもとに, デバイスが復旧した際に中断された箇所から更新を再開することが可能となる. 評価実験ではアップデートの途中で IoT デバイスに強制的に再起動を行い, 通信障害によりサーバーとの通信を中断し, 更新が途切れる状況を再現した. 提案の適用前と適用後で, 最初から更新を行う場合に要する時間を比較した. 実験では, 1 台の IoT デバイスに 4 つの更新ファイルを順次送信し, 各ファイルで 10 回ずつ中断を発生させた結果を平均値として算出した. 送信したのは, main.py の約 7.67[KB], Sensor.py の約 2.03[KB], Module.py の約 4.02[KB], Setting.py の約 0.14[KB] の 4 つのファイルである. main.py での中断時における更新時間は, 提案の適用前で約 2.9 秒, 適用後で約 4.2 秒となり, 約 45[%] 増加した. Sensor.py での中断時における更新時間は, 提案の適用前で約 4.7 秒, 適用後で約 4.5 秒となり, 約 6[%] 短縮された. Module.py での中断における更新時間は, 提案の適用前で約 4.9 秒, 適用後で約 4.7 秒となり, 約 6[%] 短縮された. Setting.py での中断時における更新時間は, 提案の適用前で約 4.9 秒, 適用後で約 4.4 秒となり, 約 11[%] 短縮された. アップデートが中断なく正常に行われる場合での更新時間は通常の更新で約 2.0 秒, 提案を含んだ更新で約 4.7 秒となり, 約 138[%] 増加した. 更新に要する時間が増加した理由は, 提案ソフトウェアの処理が実行されたためである.

1. はじめに

背景

IoT (Internet of Things) はモノのインターネットと訳され, 物理的なデバイスがインターネットを介して接続しデータの収集を行う仕組みである. IoT の利用分野としては, 例えばスマートシティ, ヘルスケア, 農業, 製造業があげられる [1–4]. これらの分野での IoT 活用方法として, 特に産業分野では IIoT (Industrial Internet of Things) と呼ばれている [5]. IIoT の実装目的としては, 企業の生産性向上, セキュリティ強化, 効率化が例として挙げられる [6]. IIoT システムでは IoT の信頼性と機能の障害や故障を考慮する必要がある [7]. 安全性は, 産業における不確実な障害を回避し, Industrial4.0 の実現において重要な役割を果たしている [8].

IoT 技術は進化が速いことから, セキュリティ更新, バグ修正, 機能の追加のための無線更新をサポートする必要性が高まっている [9]. IoT デバイスには安全な更新メカニズムが組み込まれていることが少なく, セキュリティの脆弱性を修正できないことがある [10]. そのため, 永久的なバグを抱えたままの運用になる. IoT システムは長期的な保守や改善のために更新する必要がある [11–13].

IoT デバイスのアップデートにおいて, ネットワーク接続の切断やデバイスの障害が原因で一部の更新データが適用されない場合, システムの状態が古いものと新しいものが混在する不整合が発生することがある. 例えば, デバイスがバッテリー切れや電源障害での停止, ネットワーク障害

[1] 東京工科大学コンピュータサイエンス学部
クラウド・分散システム研究室
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学大学院バイオ・情報メディア研究科
コンピュータサイエンス専攻
クラウド・分散システム研究室
〒 192–0982 東京都八王子市片倉町 1404-1

でサーバーとの通信が切断された際に更新に失敗することがある．この不整合によって，センサーデバイスのデータ取得や送信の失敗，システム全体の正常な動作を阻害する．

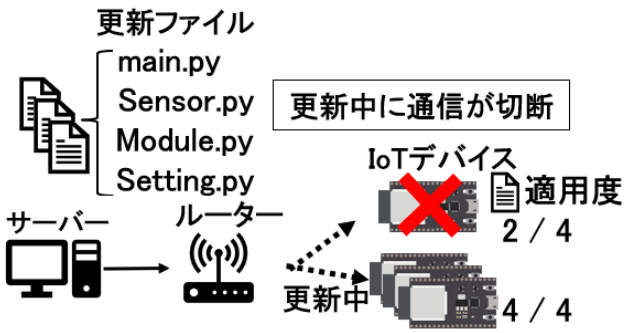

図 1: アップデート中に切断された状況

図 1 は IoT デバイスが遠隔で更新している際に，ネットワーク接続が切断された状況を示したものである．右側の適用度は更新対象のファイルが何個適用されたかを表している．更新ファイルが全て適用される場合，適用度は 4/4 になる．しかし，ネットワーク障害により一部のデバイスはサーバーとの通信が途切れることで更新プロセスが中断される．この場合，一部のファイルのみが更新された状態となり，適用度は 2/4 になる．IoT のマイクロコントローラは，静的な動作を適用されることがよくある [14]．この場合，IoT デバイスは再接続後に最初からファイルのダウンロードと適用を実行する．

課題

課題は，中断された更新プロセスにおいて，最初から更新を再開する場合，全体の更新に要する時間が増加することである．

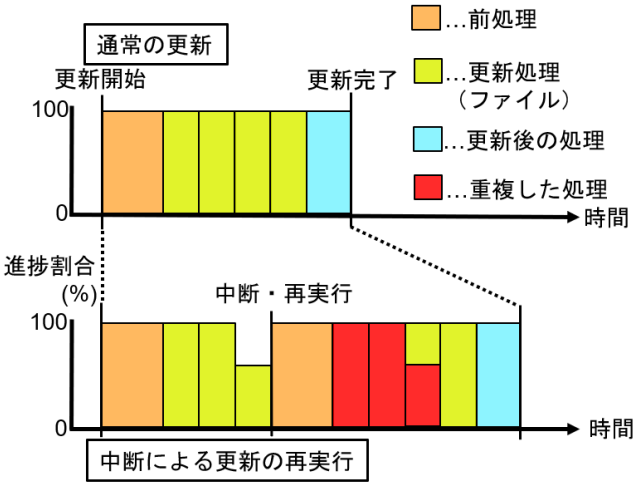

図 2: 通常の更新と中断による更新の再実行の比較

図 2 は通常の更新に要する時間と，最初から更新を行った場合に要する時間を比較したものである．前処理は Wi-Fi 接続やサーバーとの接続，更新の確認を行う処理を表している．更新処理はサーバーから受信した更新ファイルを適用する過程を表している．更新後の処理は更新完了後に IoT デバイスのバージョンをサーバーに送信する処理や，通常動作への移行を表している．また，前処理および更新後の処理は更新プロセスにおいて必要な処理であるため，これらは重複した処理には含まないものとする．図 2 の下部の中断による更新の再実行は更新処理の途中で更新が中断される場合を想定している．この場合，再接続後に更新が初めから適用されるため，図の赤い四角の部分が重複処理となる．この重複処理により，アップデートに要する時間が延長されるだけでなく，エネルギー消費量の増加やデバイスの利用停止時間の増加を招く．したがって，中断時に更新の再実行を行った場合の重複した処理を排除し，更新に要する時間を短縮するための手法が求められる．

基礎実験

基礎実験として，通常時のアップデートに要する更新時間と，アップデートが完了する直前で中断し更新を再実行した場合に要する更新時間を調査した．

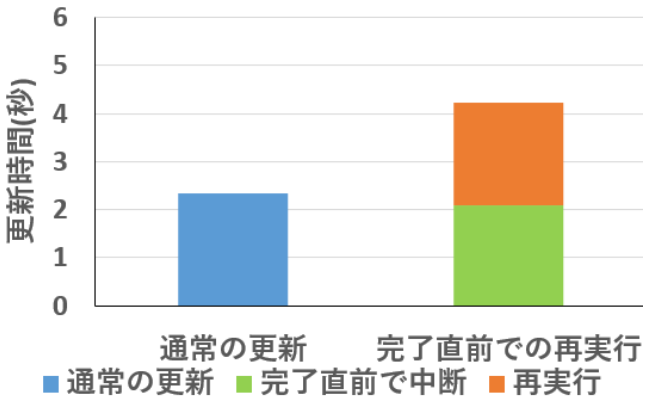

図 3: 通常時の更新と完了直前の再実行の実行結果

図 3 は通常時の更新時間と，アップデートが完了する直前で中断を行い，更新の再実行を行った際の更新時間である．通常時の更新に要する時間は約 2.3 秒であった．一方，更新が完了する直前に更新の再実行に要した時間は約 4.2 秒であった．この結果から，中断から更新の再実行を行う事で更新に要する時間が約 2 倍になる事が分かった．

各章の概要

第 2 章では，関連研究について記述する．第 3 章では提案方式の説明，ユースケース・シナリオについて記述する．第 4 章では提案方式の実装方法について記述する．第 5 章では評価実験と基礎実験について記述する．第 6 章では，本研究についての議論を記述する．第 7 章では，本研究のまとめを記述する．

2. 関連研究

ワイヤレスネットワークでのファームウェア更新中に発生する通信エラーや中断が IoT デバイスを使用不能な状態になるリスクを解決する必要がある．これに対し, OTA (Over-the-Air) プログラミング中の信頼性とエラー回復に焦点を当てている研究がある [15]．この研究では, 更新失敗時にデバイスが以前の安定した状態にロールバックできるフェールセーフ機構を提案しており, ネットワークの安定性と機能性を維持することを目的としている．二重メモリ構造を活用し, アップデートが完了するまでファームウェア更新を適用しないことで, 更新中の障害や中断が発生しても, 以前の安定版ファームウェアにロールバック可能になる．次に更新プロセスの進行状況を追跡し, 再接続時に中断された箇所から更新を再開し, 差分更新を利用して再送データを最小限に抑える．これにより, IoT デバイスの更新中に障害が発生しても安全に回復できるだけでなく, 中断された更新も可能となり, 効率的で信頼性の高い更新プロセスを実現できる．しかし, この研究はファームウェアアップデートに焦点を当てており, プログラムファイルの更新に関することは言及されていないため, この提案を適用することができない．

IoT デバイスは数が増えるにつれ, 機能の追加やバグの修正のためにファームウェアやソフトウェアを更新する事がある．コストのかかるリコールや現場での修理を避けるには, シンプルで便利かつ, 信頼性の高いワイヤレスメソッドが重要である．この課題に対し, IoT デバイスのファームウェアアップデートの管理のためにシステムを提案している研究がある [16]．クラウドベースの RESTful Web サービスを利用して, デバイスの管理を行う事を目的としている．デバイスとクラウド間での通信のために RESTful API を使用するアーキテクチャを構築する．これにより, デバイスがクラウドサービスにアクセスし, 最新のファームウェアを取得できるようになる．また, 複数のデバイスのファームウェアを同時に管理できるようにするために, アップデートのスケジューリングや失敗時の再試行の方法が提案されている．しかし, この提案はファームウェアアップデートに対する提案であり, プログラムファイルの更新に関することは言及されていない．また, この研究は更新時の失敗においてロールバックを採用しており, 本稿の課題である重複した処理の排除を行う事ができない．

3. 提案方式

本稿では, 更新プロセスで実行中の関数名やファイル名を記録し再接続時に参照することで, アップデートが中断された箇所から再開する仕組みを提案する．この手法は, 更新が中断され, 最初から更新を行うのに要する時間の短縮を目的としている．

提案における IoT デバイスおよびサーバーの詳細な条件と設定を説明する．IoT デバイスのアップデート対象は温度データを取得する機能を有する Sensor.py および Module.py である．これらに気圧データを取得する機能の追加を行う．また, 送信情報の設定や既存ファイルのパッチ適用を目的に main.py および Setting.py についても更新を行う．更新対象のプログラムファイルは main.py, Sensor.py, Module.py, Setting.py とし, この順で更新を行う．アップデートは進行状況をリアルタイムで監視しやすくするため, IoT デバイスのような限られたリソース環境に適したセグメント方式を採用する．アップデート処理は通常, IoT デバイスからサーバーにリクエストを送信し, データを受信する方式で実施される [16]．したがってクライアント駆動型で行う．

本稿の提案では, IoT デバイス側で更新プロセスの記録と読み込みを行い, サーバー側で送信されたファイルの記録を利用することで, デバイスごとの中断された箇所からの更新の再実行を行う仕組みを構築する．提案手法の詳細を, IoT デバイスとサーバーの 2 つの観点から分けて説明する．

IoT デバイス

IoT デバイスでは更新プロセスで実行している関数名をファイルに記録する．また, 現在行っている処理や更新中のファイル名を記録するために, テキストフォーマット形式のファイルを作成して保存する．関数の処理が終了した場合, 次の関数を実行し, 記録内容を上書する．更新ファイルを新たに受信した際にファイル名の記録を上書きする．この仕組みにより, 更新のプロセスが中断しても, サーバーに再接続後にテキストフォーマット形式のファイルを参照することで, 未完了部分の関数から再実行が可能となる．

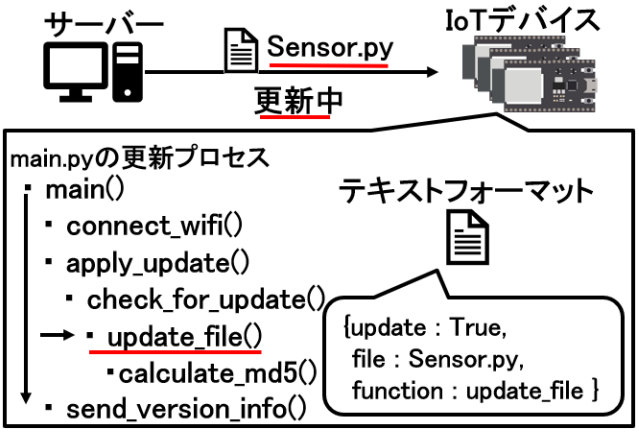

図 4: 更新中の IoT デバイスでの動作

図 4 は IoT デバイスが更新中の動作を示したものである．提案を説明するために 4 つのプログラムファイルを更新する状況を例として挙げる．ファイル名は main.py, Sensor.py, Module.py, Setting.py とする．ここでは, IoT

デバイスが更新ファイルの 2 番目である Sensor.py の更新を行っていると想定する．その際，テキストフォーマット形式のファイルを作成し，更新の最中であるのか，更新中の Sensor.py，および基幹プログラムの main.py で処理を行っている関数名をファイルに記録する．この記録をもとにサーバーに再接続後，中断された箇所から更新の再実行を行う．また，受信した Sensor.py を IoT デバイスの一時ファイルに保存することで再接続時に，Sensor.py の再要求と再受信を行わずに，更新ファイル受信後の処理から再開することで，重複処理を排除する．更新完了の判定は，更新プロセスが終了した際に，デバイスが保持しているファイル名とバージョンを記述したメタデータをサーバーに送信することで行う．その後，サーバーからの受信が完了し，更新完了の応答を得た場合に，デバイスの標準動作へ移行する．この仕組みにより，重複処理を排除し，中断された箇所から更新の再開が可能となる．

サーバー

サーバーでは IoT デバイスが初回の接続か，更新途中であるかを確認を行う．更新プロセスを開始してから初回接続の場合，通常通りに更新を行う．一方，2 回目以降に接続され，更新途中である場合は，どの更新ファイルを送信しているかの確認を行い，IoT デバイスからのリクエストを待機する．

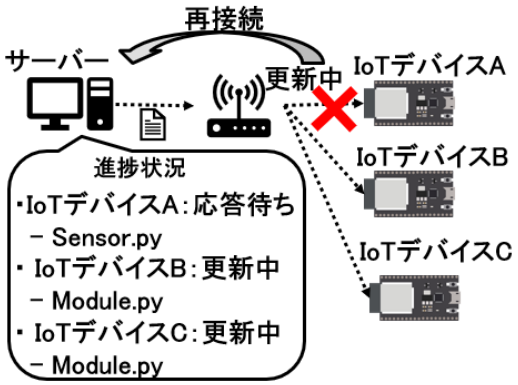

図 5: 更新中のサーバーでの動作

図 5 は更新中のサーバー側の動作を示したものである．サーバーは，IoT デバイスごとのバージョンの管理を行うために，各デバイスに固有の ID を割り当てている．また，デバイスからのリクエストに応じて更新する各ファイル名．バージョン，ハッシュ値を含むメタデータと更新ファイルを送信する．図 5 では IoT デバイス A, B, C が個別に割り当てられた ID を持つと仮定する．サーバーは，これらのデバイスの更新状況を管理する．この状況下で，各デバイスにアップデートを行い，デバイス A のみ更新が中断された場合を想定する．デバイスが再接続後に，送信済みのファイルを確認し，未送信のファイルから送信を再開する．この仕組みにより，IoT デバイスは更新ファイルを最初からではなく，中断された箇所から受信することが可能となる．また，各 IoT デバイスの更新が終了した際にファイル名とバージョンを含むメタデータを受信し．全ての IoT デバイスのメタデータを受信することでサーバー側で更新が完了したと判定する．

ユースケース・シナリオ

本稿の提案は，産業用センサーにアップデートを適用する場面を想定している．

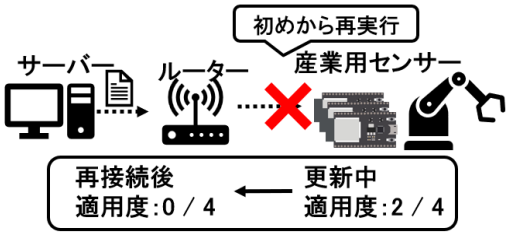

図 6: ユースケース・シナリオ

図 6 にユースケース・シナリオを示す．産業用センサーでデータの収集を行うために IoT のセンサーデバイスをもちいている．機能の追加やパッチの適用のために，アップデートを行うが，デバイスの停止によりネットワーク接続が切断され更新プロセスが中断される．デバイスは再起動後，サーバーと接続するが，中断された更新を初めから行うと，更新に要する時間が増加する．これは迅速な更新やデータの処理が遅れや生産性の低下を引き起こす．本提案を適用することで，再接続と同時に中断されたプロセスから再開することができるため，更新時間の短縮が可能となる．

4. 実装

IoT デバイスおよびサーバーの実装について説明する．本稿の提案ソフトウェア構成を図 7 に示す．

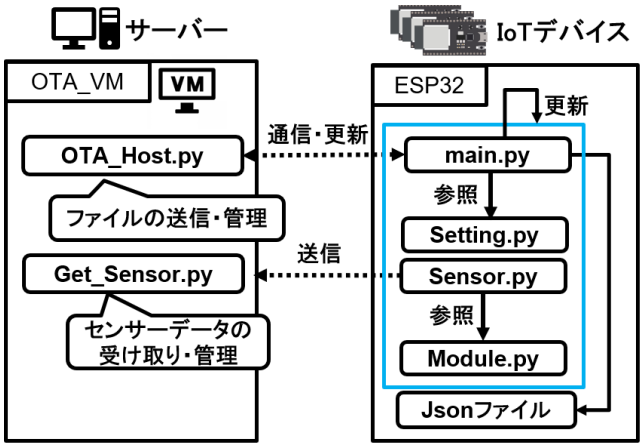

図 7: ソフトウェア構成図

サーバーでは『OTA_VM』と名付けられた仮想マシン環境を構築し，2 つの HTTP サーバーを Python で実装

した．また, IoT デバイスごとの更新ファイルの管理を行うために, version.json を作成した．『OTA_Host.py』は Python の SimpleHTTPRequestHandler モジュールを使用して HTTP サーバーを構築し, IoT デバイスとの通信を行いアップデートを行うためのプログラムである．『Get_Sensor.py』は Python の BaseHTTPRequestHandler モジュールを使用して HTTP サーバーを構築し, IoT デバイスからセンサーデータを受信および管理するためのプログラムである．

IoT デバイスには ESP32 を使用し, 4 つのプログラムと 1 つのテキストフォーマットファイルを MicroPython で実装した．『main.py』はサーバーとの通信, 更新, その他プログラムを動作させるための基幹プログラムである．『Setting.py』は Wi-Fi の設定および各種機能の設定を記述したプログラムである．『Sensor.py』は I2C バスの初期化を行い, Module.py を参照しセンサーデータの取得とサーバーへデータの送信を行うプログラムである．『Module.py』はアドレスおよびレジスタの設定, センサーの初期化, データ処理を行うプログラムである．『function-saved.json』は現在の更新状況, 更新中のファイル名, 実行中の関数名を記録したテキストファイルである．提案ソフトウェアは main.py で動作する．更新プロセス中に実行された関数名や更新中のファイル名, 更新中であるのかを function-saved.json に保存する．プロセスが再開された場合, 更新が中断された状態であれば更新中のファイル名と実行中の関数名を参照し, 実行途中の関数から更新を再実行する．

アップデートは IoT デバイスが定期的にサーバーにある OTA_VM にある OTA_Host.py で構築した HTTP サーバーにリクエストを送信することで開始される．サーバーがリクエストを受信すると, レスポンスとして更新するファイル名と各ファイルのハッシュ値を送信する．その後, センサーデータの送信を一時的に停止し, 更新対象である main.py, Sensor.py, Module.py, Setting.py の各ファイルを順次送信し更新を行う．更新完了後, Sensor.py は OTA_VM 内の Get_Sensor.py で構築した HTTP サーバーにセンサーデータを送信する．

5. 評価実験

評価実験として, アップデートの途中で IoT デバイスに強制的に再起動を行い, 通信障害によりサーバーとの通信を中断し, 更新ファイルが一部適用されていない状況を再現した．停止するタイミングは, IoT デバイスが main.py, Sensor.py, Module.py, Setting.py の各更新ファイルを受信した時点とした．このタイミングで停止させる理由は, 本稿の目的が重複した処理を回避し, 最初から更新を実行した場合に要する時間を短縮することにあるためである．受信した更新ファイルを保存し, サーバーに接続した際に, 再度要求を行わないために, この時点でデバイスを停止させた．1 台の IoT デバイスに 4 つの更新ファイルを送信し, 提案の適用前と適用後で, 各ファイルを受信した際に中断をそれぞれ 10 回実施した．未更新ファイルが 1 つから 4 つの場合において, 各更新ファイルの中断までに要する時間と, 最初から更新を実行した場合で更新プロセスが完了までの全体の時間を測定し, これを平均値として表し比較した．

実験環境

実験は以下の環境で実施する．

IoT デバイス

- ESP32
  Espressif Systems 社が提供する Wi-Fi や Bluetooth 機能を持ち合わせた低消費電力のマイクロコントローラである [17]．本稿では IoT デバイスとして ESP32 を使用した．
- BMP280
  Bosch Sensortec 社が開発した環境センサーである．温度と気圧を測定することができ, これらの測定のために BMP280 センサーは広く使用されている [18]．本稿ではアップデートを行った際に, センサーが取得するデータが追加されているかの確認を行うために使用した．
- MicroPython
  Python 言語をベースにした軽量なプログラミング言語である．ESP32 の制御プログラムの制作に使用した．

サーバー

- 仮想マシン (Virtual Machine)
  IoT デバイスとの通信, 更新, データの受信を行うためのサーバー環境を構築するために, Ubuntu 24.04 LTS をインストールした仮想マシンを用意した．
- Python SimpleHTTPRequestHandler
  ESP32 に遠隔更新を行うために使用した．更新を行うためのサーバーには Python のモジュールの 1 つである SimpleHTTPRequestHandler を使用し, 更新ファイルの配布と管理, 受信データの管理を行った．
- Python BaseHTTPRequestHandler
  ESP32 からセンサーデータを受け取るために使用した．センサーデータを受け取るためのサーバーには Python モジュールの 1 つである BaseHTTPRequestHandler を使用し, デバイスごとのデータの受け取りと管理を行った．

更新ファイル

実験でもちいた更新ファイルのサイズを表 1 に示す．main.py は約 7.67[KB], Sensor.py は約 2.03[KB], Module.py は約 4.02[KB], Setting.py は 0.14[KB] である．これらを IoT デバイスに送信する更新ファイルとし, 機能の追加とパッチ適用を行うアップデートを実施した．

表 1: サーバーが IoT デバイスに送信する更新ファイル

ファイル名	サイズ [KB]
main.py	約 7.67
Sensor.py	約 2.03
Module.py	約 4.02
Setting.py	約 0.14

実験結果と分析

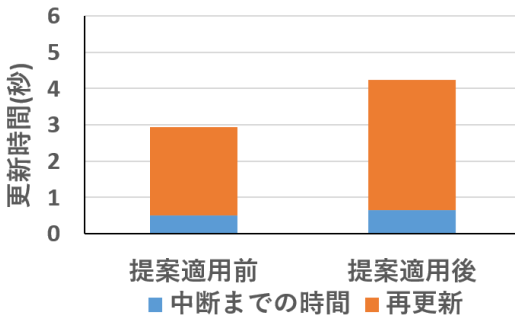

図 8: main.py で中断した実行結果

図 8 は, 更新ファイルの 1 つ目である mian.py を受信し中断した際の実行結果を表している. 提案の適用前で更新に要する時間は平均約 2.9 秒であり, 適用後は平均は約 4.2 秒となった. 全体の更新に要する時間は約 45%増加した. 中断までの時間は提案の適用前と適用後で平均約 0.5 秒から平均約 0.6 秒に増加し, 最初から更新を実行した場合に要する時間は平均約 2.4 秒から平均約 3.5 秒に増加した. この結果から, 中断後の再実行時においても, 提案プログラムが実行されることで更新プロセスに必要な処理が増加したことで, main.py の中断の場合には全体の更新に要する時間が増加したことが分かる.

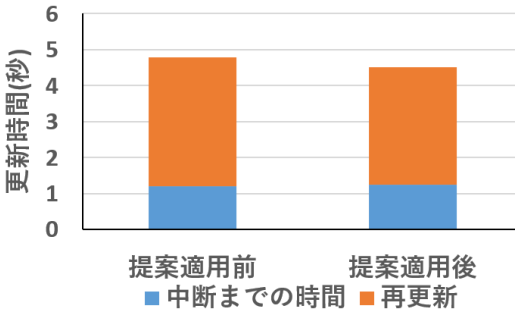

図 9: Sensor.py で中断した際の実行結果

図 9 は更新ファイルの 2 つ目である Sensor.py を受信し中断した際の実行結果を表している. 提案の適用前で更新に要する時間は平均約 4.7 秒であり, 適用後は平均約 4.5 秒となった. 全体の更新に要する時間は約 6[%] 短縮された. 中断までの時間は提案の適用前と適用後で平均約 1.1 秒から平均約 1.2 秒に増加し, 最初から更新を実行した場合に要する時間は平均約 3.5 秒から平均約 3.2 秒に短縮された. この結果から, 中断までの時間は約 0.1 秒増加し, 最初から更新を実行した場合に要する時間は約 0.2 秒短縮されたことが分かる. 全体の更新に要する時間は約 0.2 秒短縮された.

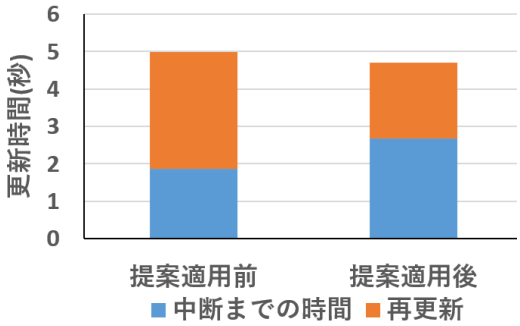

図 10: Module.py で中断した際の実行結果

図 10 は更新ファイルの 3 つ目である Module.py を受信し中断した際の実行結果を表している. 提案の適用前に更新に要する時間は平均約 4.9 秒であり, 適用後は平均約 4.7 秒となった. 全体の更新に要する時間は約 6[%] 短縮された. 中断までの時間は提案の適用前と適用後で平均約 1.8 秒から平均約 2.6 秒に増加し, 最初から更新を実行した場合に要する時間は平均約 3.1 秒から平均約 2 秒に短縮された. この結果から, 最初から更新を実行した場合に要する時間は短縮されたが, 中断までの時間が増加したことで, 提案の更新プロセスの記録を行う処理が, 実行時間を増加させたことが予想される.

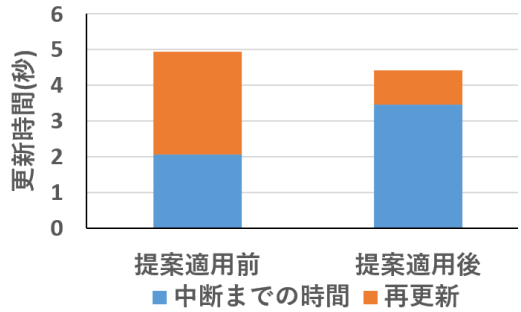

図 11: Setting.py で中断した際の実行結果

図 11 は更新ファイルの 4 つ目である Setting.py を受信し中断した際の実行結果の結果を表している. 提案の適用前で更新に要する時間は平均約 4.9 秒であり, 提案適用後は平均約 4.4 秒であった. 全体の更新に要する時間は約 11[%] 短縮された. 中断までの時間は, 提案の適用前で平均約 2.0 秒, 適用後で平均約 3.4 秒に増加し, 最初から更新を実行するのに要する時間が平均約 2.8 秒から約 0.9 秒に短縮された. この結果, Module.py の場合と同様に, 提案の記録を行う処理が更新プロセスの複雑性を高め, 全体の更新時間を増加させた. しかし, 最初から更新を実行した場合に要する時間は大幅に短縮された.

提案を実装したプログラムにおいては, 更新に要する時間が増加した. そのため, 更新プロセス中の中断からの再処理を行わない場合で更新に要する時間がどの程度増加するのかについて調査を実施した.

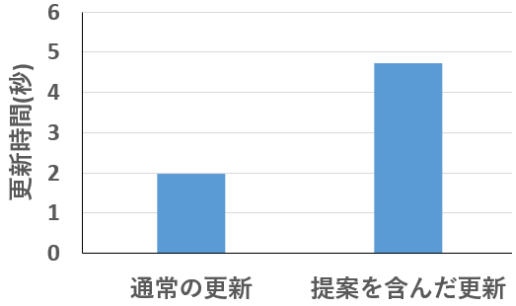

図 12: 更新プロセス中の中断が発生しない場合での更新時間の比較

アップデートが

図 12 は提案の適用前と適用後で更新プロセスが中断なく正常に行われる場合で, 1 回のアップデートで全ての更新ファイルを適用する際に要する時間を比較したものである. 提案適用前の更新プログラムでは平均約 2 秒であり, 提案適用後のプログラムでは平均約 4.7 秒であった. 全体の更新に要する時間は約 138[%] 増加した. この結果から, 提案を適用した場合, 1 回のアップデートに要する時間が平均約 2.7 秒増加することが分かった.

以上のことから, 提案をもちいることで 1 回の更新に要する時間は, 更新ファイルの数が増加するほど処理が削減され, 時間を短縮できることが分かった. 最大で 4 つ目の Setting.py での中断時において更新時間は, 提案の適用前と適用後で, 平均約 4.9 秒から平均約 4.4 秒に短縮され, 全体の更新に要する時間は約 11[%] 短縮された. しかし, 提案による更新プロセスの記録は, プロセスの複雑性を高める結果となり, 各ファイルの更新に要する時間が増加することが分かった. 更新処理の複雑化と更新の再実行に要する時間の短縮にはトレードオフの関係があることが予想される. また, 全ての結果において提案適用前の更新の再実行は, 図 12 の通常の更新と同様の処理を行っているため, 本来であれば同じ処理時間となるはずである. しかし, 図 9 の Sensor.py, 図 10 の Module.py, 図 11 の Setting.py では提案適用前の更新の再実行に要する時間が大幅に増加した. この要因として, ESP32 がファイルの更新中に処理負荷の増大による一時的な処理の停止が発生したことや, サーバーからのファイル受信において遅延が発生したことが挙げられる.

6. 議論

本稿では, 更新プロセス中の実行している関数名と更新中のファイル名の記録を利用することで, 中断された箇所から更新を再開する手法を提案した. 提案手法では, 更新の中断された時点の関数から処理を再開することで, 最初から更新を実行した場合に要する時間を短縮できた. しかし, 1 回のアップデートが完了するまでに要する時間が平均で約 2.7 秒増加し, 全体の更新に要する時間は約 138[%] 増加した. この要因として, 更新を再開する際に, 更新ファイルを受信するたびに一時ファイルに保存する処理や, 実行中の関数名および更新中のファイル名を記録する処理が必要となることが挙げられる. この結果, 関数名やファイル名の記録を行うファイルへの書き込みや読み込みの処理が更新プロセスを複雑化させたことが主な原因である. この課題を解決する方法として, 更新ファイルの一時ファイルへの保存を行わず, 受信した更新ファイルをメモリで保持し, 再接続後に失った更新ファイルを要求することを前提とすることが挙げられる. この場合, 更新プロセスの中断された箇所に基づく処理の再開は可能であるものの, 更新を中断時から再開する際に要する時間は本稿の実験結果よりも増加すると予測される.

提案手法では, 更新プロセスにおいて, 中断前および中断後の更新の再実行においても, 常に提案の更新ファイルの一時ファイルへの保存処理, 実行中の関数名および更新中のファイル名を記録する処理が実行される. このため, 図 8 で示されているように, 1 つ目の main.py で中断した場合, 更新の再実行後においても提案手法による更新プロセスの記録処理が実行される. このため, 提案の適用前よりも適用後の更新に要する時間が増加してしまう. この課題を解決する方法として, 一部の更新プロセスの再実行時において, 提案手法を適用させないことが挙げられる. 例として, 図 8 の実験結果では, 最初から更新を実行した場合に要する時間は, 提案の適用後では約 3.5 秒であるが, 適用前では約 2.4 秒である. したがって提案を適用しないことで更新に要する時間の増加を防ぐことができる. 本稿の実験結果からは main.py のみで更新プロセスの再実行時に提案手法を適用しないことが有効であると判断できる. しかし, 他の更新ファイルにおいて, 更新中に処理負荷の増大による一時的な処理の停止が発生や, サーバーからのファイル受信において遅延が発生することがある. この場合, 提案手法を適用後, 更新の再実行に要する時間が増加する. そのため, どの更新ファイル受信時に提案手法を適用するか, または適用をしないのかの判断を行うために, 再実験が必要である.

7. おわりに

課題は IoT デバイスをアップデートする際, ネットワーク障害によりサーバーとの通信が切断されることで更新プロセスが中断され, 一部のプログラムが適用されないことである. デバイスとサーバーのネットワーク接続が復旧した場合, 更新プロセスを最初から行うのでは更新に要する

時間が延長され, ネットワーク帯域やエネルギーを浪費される. 提案では, 更新プロセス中の処理を実行している関数名や現在更新を行っているファイル名を記録し, デバイスが起動時に記録さをもとに中断された箇所からの再開手法を提案した. 評価ではアップデートプロセス中に IoT デバイスを更新ファイルごとに停止させ, 中断に要する時間と最初から更新した場合に要する時間を測定し, 1 回のアップデートに要する時間を評価した. 実験結果として, 一部の条件を除いて再行使に要する時間は短縮されたが, 1 回のアップデートに要する時間は増加した.

謝辞

本稿の執筆にあたり, ご指導, ご意見を頂きました, 東京工科大学コンピュータサイエンス学部の山本真也さん, 越後谷澪さん, 筒井優貴さんに御礼申し上げます.

参考文献

[1] Mocnej, J., Pekar, A., Seah, W. K., Papcun, P., Kajati, E., Cupkova, D., Koziorek, J. and Zolotova, I.: Quality-enabled decentralized IoT architecture with efficient resources utilization, *Robotics and Computer-Integrated Manufacturing*, Vol. 67, p. 102001 (2021).

[2] Alamri, M., Jhanjhi, N. and Humayun, M.: Blockchain for Internet of Things (IoT) research issues challenges & future directions: A review, *Int. J. Comput. Sci. Netw. Secur*, Vol. 19, No. 1, pp. 244–258 (2019).

[3] Thakkar, A. and Lohiya, R.: A review on machine learning and deep learning perspectives of IDS for IoT: recent updates, security issues, and challenges, *Archives of Computational Methods in Engineering*, Vol. 28, No. 4, pp. 3211–3243 (2021).

[4] Meneghello, F., Calore, M., Zucchetto, D., Polese, M. and Zanella, A.: IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices, *IEEE Internet of Things Journal*, Vol. 6, No. 5, pp. 8182–8201 (2019).

[5] Gebremichael, T., Ledwaba, L. P., Eldefrawy, M. H., Hancke, G. P., Pereira, N., Gidlund, M. and Akerberg, J.: Security and privacy in the industrial internet of things: Current standards and future challenges, *IEEE Access*, Vol. 8, pp. 152351–152366 (2020).

[6] Shakya, S. R. and Jha, S.: Challenges in Industrial Internet of Things (IIoT), *Industrial Internet of Things*, CRC Press, pp. 19–39 (2022).

[7] : Guest Editorial: Trustworthiness in Industrial Internet of Things Systems and Applications, *IEEE Transactions on Industrial Informatics*, Vol. 16, No. 9, pp. 6079–6082 (online), DOI: 10.1109/TII.2020.2983387 (2020).

[8] Kaur, G. and Chanak, P.: An Intelligent Fault Tolerant Data Routing Scheme for Wireless Sensor Network-Assisted Industrial Internet of Things, *IEEE Transactions on Industrial Informatics*, Vol. 19, No. 4, pp. 5543–5553 (online), DOI: 10.1109/TII.2022.3204560 (2023).

[9] Bauwens, J., Ruckebusch, P., Giannoulis, S., Moerman, I. and De Poorter, E.: Over-the-air software updates in the internet of things: An overview of key principles, *IEEE Communications Magazine*, Vol. 58, No. 2, pp. 35–41 (2020).

[10] Zandberg, K., Schleiser, K., Acosta, F., Tschofenig, H. and Baccelli, E.: Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check, *IEEE Access*, Vol. 7, pp. 71907–71920 (online), DOI: 10.1109/ACCESS.2019.2919760 (2019).

[11] Villegas, M. M. and Astudillo, H.: OTA updates mechanisms: a taxonomy and techniques catalog, *XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIO 49 (Modalidad virtual)* (2020).

[12] Zhang, C., Ahn, W., Zhang, Y. and Childers, B. R.: Live code update for IoT devices in energy harvesting environments, *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, IEEE, pp. 1–6 (2016).

[13] El Jaouhari, S.: Toward a Secure Firmware OTA Updates for constrained IoT devices, *2022 IEEE International Smart Cities Conference (ISC2)*, pp. 1–6 (online), DOI: 10.1109/ISC255366.2022.9922087 (2022).

[14] Zyrianoff, I., Sciullo, L., Gigli, L., Trotta, A., Kamienski, C. and Di Felice, M.: An Over the Air Software Update System for IoT Microcontrollers based on WebAssembly, *2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pp. 331–338 (online), DOI: 10.1109/DCOSS-IoT61029.2024.00057 (2024).

[15] Unterschütz, S. and Turau, V.: Fail-safe over-the-air programming and error recovery in wireless networks, *Proceedings of the 10th International Workshop on Intelligent Solutions in Embedded Systems*, pp. 27–32 (2012).

[16] Sun, C., Xing, R., Wu, Y., Zhou, G., Zheng, F. and Hu, D.: Design of Over-the-Air Firmware Update and Management for IoT Device with Cloud-based RESTful Web Services, *2021 China Automation Congress (CAC)*, pp. 5081–5085 (online), DOI: 10.1109/CAC53003.2021.9727516 (2021).

[17] Maier, A., Sharp, A. and Vagapov, Y.: Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things, *2017 Internet Technologies and Applications (ITA)*, IEEE, pp. 143–148 (2017).

[18] Kusuma, H. A., Oktavia, D., Nugaraha, S., Suhendra, T. and Refly, S.: Sensor BMP280 Statistical Analysis for Barometric Pressure Acquisition, *IOP Conference Series: Earth and Environmental Science*, Vol. 1148, No. 1, IOP Publishing, p. 012008 (2023).

