稼働条件を満たさないIoTデバイスにおけるソフトウェアのバージョン比較を用いた更新するファイルの判断と適用

宮本 港斗[1]　大沢 恭平[2]　串田 高幸[1]

概要：IoT 技術の普及により家電やスマートフォンに留まらず幅広い分野で活用されている．IoT デバイスはバグの修正や機能追加のためのソフトウェアのアップデートを行う必要がある．しかし, その中でバッテリー駆動ではない AC アダプターや PoE から電源を供給されている IoT デバイスは, 更新の最中に停電障害によりサーバーとの通信が切断され, アップデートが完全に適用されずデバイスが正常に動作しなくなる．本稿では, 障害が発生し不完全なアップデートになった際, デバイス側のプログラムのバージョンが稼働条件に満たないもののみを再送信することを提案し, 迅速な障害復旧を目的とする．デバイスが障害から復帰時にサーバーと通信し, クライアントのバージョン情報とサーバーのバージョン管理ファイルを比較する．その際に, 稼働条件に満たないバージョンがある場合, サーバーは再送信が必要な対象の更新ファイルを送信する．基礎実験ではアップデートが正常に行われた状態とアップデート中に通信を切断した状態の比較を行った．切断時について評価を行い, 送信されるセンサーデータは更新前と変わらなかったため, アップデートが中断されたデバイスは管理者が想定した動作にならない．評価実験では, アップデート中に通信を切断した際に更新されていないファイルと稼働条件に満たないファイルの検出率と, アップデートを最初から行った場合と再送信した場合で復旧に要した時間がどれほど削減されたかを評価する．

1. はじめに

背景

IoT とは”Internet of Things”の略でモノのインターネットと訳される．これはインターネットを通じて物理的なデバイスが相互に接続され, データの収集や交換を行うシステムを指す．IoT の用途の一例として, スマートシティ, ヘルスケア, 農業, 製造業がある [1–3]．このような産業における IoT は, 産業用 IoT(IIoT = Industrial Internet of Things) と呼ばれる [4]．産業用 IoT を実装するのには, 企業の生産性, セキュリティ, 効率性の向上といった目的がある [5]．IoT 技術は急速に成長しており生活の質の向上に大きく貢献している [6]．IoT は, インターネットを通じて多種多様なデバイスが相互に接続されることで, 情報の収集や自動化を実現している．そのため, IoT 技術を活用し個人の生活から産業まで, 幅広い分野で効率化と利便性の向上が図られている．身近なモノの例では, IoT ウェアラブルデバイスとして着用することで利便性が向上したり, スマート家電として利用されることで効率的な生活が実現されている [7]．IoT は現代のトレンドであるだけでなく, 日常生活に統合できる手頃なソリューションでもある [8]．

IoT 技術は様々なシチュエーションで用いられるが, 展開時に使用されたファームウェアやプログラムを定期的な更新なしに使い続けることが適切とは限らない．他の多くのシステムと同様に, IoT デバイスもバグ修正, 機能追加, 計算能力の向上のために長期的なソフトウェアの更新が必要になる [9–11]．ほとんどの 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ないため, 重大なセキュリティの脆弱性を修正できず, 永久的な負債となる可能性がある [12]．また, 不適切なデバイス管理とファームウェア配布メカニズムは, デバイス所有者のセキュリティとプライバシーに対する多くのリスクを引き起こす可能性もある [13]．IoT の進化が速いため, セキュリティ更新, バグ修正, ソフトウェア拡張のための無線ソフトウェア更新をサポートする必要性が高まっている [14]．そのため, IoT デバイスは運用維持の為の継続的な更新メカニズムが必要である．

課題

IoT デバイスの更新に対する課題は, アップデート中の通信の切断により, プログラムの更新が完全に適用されないことである．例えば, アップデート最中に停電障害が起

[1] 東京工科大学コンピュータサイエンス学部
〒 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
〒 192–0982 東京都八王子市片倉町 1404-1

こると, 直接 AC アダプターや PoE(Power over Ethernet) から電力を供給している IoT デバイスは, インターネット通信が切断されてしまう. そのため, アップデートが中断されたデバイスは管理者が想定した動作にならない.

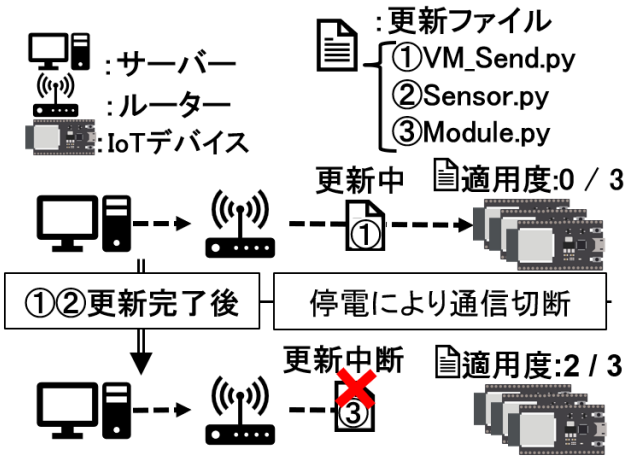

図 1: アップデートの最中に切断された状況

図 1 は IoT デバイスを遠隔で更新している最中に通信が切断された状況を示した図である. 更新ファイルは VM_Send.py, Sensor.py, Module.py の 3 種あり, これらは本稿の実験で用いたプログラムである. 第 4 章の実装にて説明を行う. 図 1 の上部のデバイスは更新の適用度が 0/3 で, VM_Send.py を送信しているためアップデートを始めた段階である. その後, VM_Send.py と Sensor.py の送信と適用が終了し, 最後の Module.py を送信している最中に停電障害が起こった. この時, デバイスに Module.py の更新が適用されていなかった. そのため, デバイスの適用度は 2/3 となり一部の更新ファイルが送信されず, 部分的な更新になる. そのため管理者が想定した動作にならない. 以上の事から, アップデート適用の最中に通信が切断された場合, 適用されなかった更新プログラムを再送信する手法が求められる.

各章の概要

第 2 章では, 関連研究について紹介する. 第 3 章では提案方式の説明, ユースケース・シナリオについて説明する. 第 4 章では提案方式の実装方法について説明する. 第 5 章では評価実験と基礎実験の結果, 評価方法について説明する. 第 6 章では, 本研究についての議論を述べる. 第 7 章では, 本研究のまとめを述べる.

2. 関連研究

クラウドサービスにおける障害復旧に関する研究を行っている [15]. 特に, 復旧メカニズムが小さな障害を拡大させ, システム全体に深刻な影響を与える可能性があることに焦点を当てている. クラウドサービスの障害復旧において「まず害をなさない」(primum non nocere) という原則を重視するアプローチを提案している. 具体的には, システムの活動状況を監視し, 安全に復旧できるタイミングを見計らって復旧作業を行うことで, 障害の拡大を防ぐ. これにより, 復旧プロセスが新たなリスクを生じさせないようにすることができる. この論文は, クラウドサービス全体の障害復旧に焦点をあてており, システム全体を監視する必要がある.

ソフトウェア定義ネットワーク (SDN) における自動障害回復メカニズムに関する研究をしている [16]. SDN は, ネットワーク制御をソフトウェアにより柔軟に管理できる技術であり, 効率的なネットワーク運用が可能でだが, 障害が発生した際の回復メカニズムが重要な課題となっている. 障害を検出すると, 障害が発生した要素を除いた新しいコントローラーインスタンスを生成し, 障害発生前に観測された入力を再生し, 障害が発生した要素を考慮したエミュレートされたネットワークの転送状態を設定しする。エミュレートされた転送状態と現在の転送状態との差分ルールセットをインストールして, ネットワークを回復する. この論文は SDN の障害復旧メカニズムに焦点をあてており, ネットワークの障害回復を自動化し, 開発者が障害に依存しないコードを記述できるようにすることを目的としている.

IoT アプリケーションにおけるフォールトトレランスメカニズムの実装に焦点を当てている研究がある [17]. 具体的には, 通信の障害やデバイスの故障といったの障害に対する柔軟な対応を可能にするプログラミングフレームワークを提案している. データの冗長性やリカバリポイントの設定、エラー検出および修復メカニズムの実装方法について述べている. これにより, アプリケーションやデバイスの状態を監視し, 障害が発生した場合に自動的にシステムを回復させるフレームワークを提案している. この論文はフォールトトレランスメカニズムに焦点を当てており, 通信障害やデバイスの故障といった復旧に関する技術的な手法を提案している.

IoT サービスの回復力を高めるためのトラフィック分割に基づくサービス埋め込みアプローチを提案している研究がある [18]. IoT の普及に伴い, 予期しない障害やサイバー攻撃, 信頼性の低い接続, 限られたリソースによる課題が増している. この研究では, トラフィック分割を用いた IoT サービス埋め込みの回復力を高める新しいアプローチを提案している. 提案されたアプローチでは、サービス指向アーキテクチャ (SOA) を活用し, 最適化モデルを用いて最適な物理リソースを決定した. モデルは混合整数線形計画法 (MILP) を用いて, 総電力消費量とトラフィック遅延を最小化する. 実験結果から, 提案アプローチにより最大 35%の電力節約と最大 37% の遅延削減が実現されたことが示された. この論文は一般的な多くの IoT ネットワーク

に対してにの提案をしており, 大規模な IoT ネットワークや動的なネットワークへの適用に対して, 拡張性と動的な対応力における評価が足りていない.

3. 提案方式

提案手法にて, 稼働条件に満たないプログラムファイルを再送信することでの早急な障害復旧を目的とする.

本稿では, 障害復旧時にサーバーが保持するバージョン管理ファイルの最低稼働条件のバージョンとデバイスが保持しているプログラムファイルのバージョンを比較し, 稼働条件に満たないプログラムファイルを検出することを提案する.

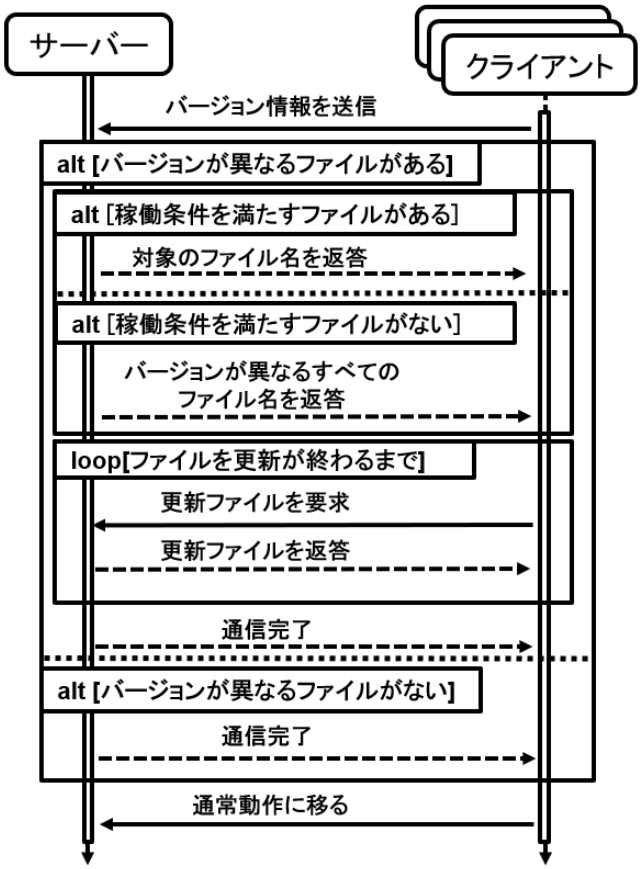

図 2: 提案方式の概要

図 2 に提案の概要を示す. 図 2 にあるように, デバイスが停電から復旧しサーバーに接続した際に, サーバーに対しクライアントが保持するバージョン情報のみを送信する. サーバー側で送られてきたバージョン情報をサーバーが保持するバージョン管理ファイルとの比較を行う. そこで, バージョンが異なるファイルがあるのか, 稼働条件を満たすファイルの有無を判定し結果に応じた返答を行う.

提案を説明するための詳細な条件や設定を説明する. IoT デバイスをアップデートする目的は, 温度データのみを取得する機能がある ver.1.0 のプログラムに, 気圧データを取得するための機能を追加した ver.2.0 に更新を行う. 更新対象のプログラムファイルは VM_Send.py, Sensor.py, Module.py とし, この順番で更新を行う. また, Sensor.py はバージョン ver.1.0 でも動作するプログラムである. 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に, IoT デバイスのような限られたリソースで用いられるセグメント方式で行うものとする. クライアントが保持するプログラムファイルのバージョン情報はプログラム内に記述してある情報を取得する.

図 3: プログラム内のバージョン情報の記載例

VM_Send.py のバージョン情報の記載例を図 3 に示す. 2 行目にファイル名, 3 行目にバージョン数のように, 2 つの要素で構成されたバージョン情報がプログラム内に記述されているものとする. 各ファイルのファイル名とプログラム内から取得したバージョン数で dict 型オブジェクトを作成し, サーバーに送信するためのバージョン情報とする.

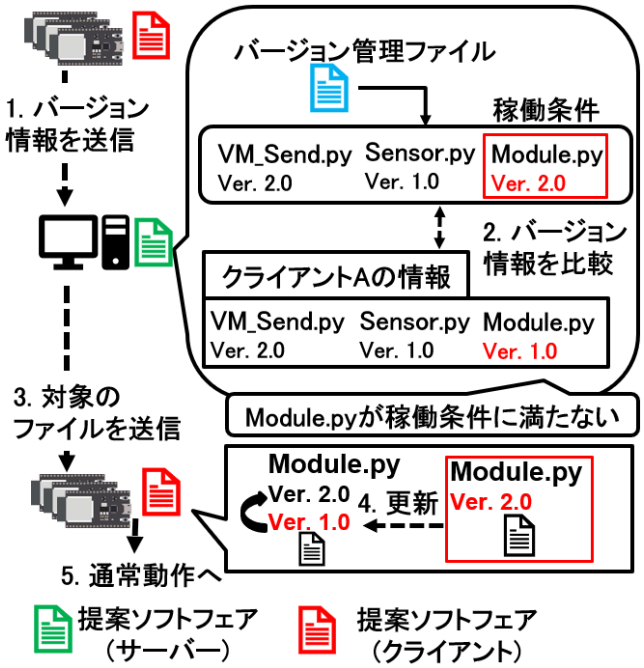

図 4: バージョン情報が異なる場合の動作

図 4 は IoT デバイスが障害から復旧した際の動作を示したものである. デバイスが起動した際, サーバーに対して更新対象である VM_Send.py, Sensor.py, Module.py のバージョン情報のみを送信する. 次にサーバー側で送られてきたバージョン情報と, サーバーが保持するバージョン管理ファイルを参照してバージョンの比較を行う. バージョン管理ファイルの記載例を図 5 に示す.

サーバー
バージョン管理ファイル

filename	latest	operational_ requirement	required
VM_Send.py	2.0	2.0	1.0
Sensor.py	2.0	1.0	1.0
Module.py	2.0	2.0	1.0

図 5: バージョン管理ファイルの記載例

図 5 のようにバージョン管理ファイルにはクライアントのプログラムが動作するためのバージョンの組み合わせが記載されている．例えばアップデートが適用された最新の状態である latest, 更新が中断された際, 障害が起こった際の稼働要件の operational_requirement, 最低限動くための必須要件の required のバージョン情報が記載されている．

バージョンは一番高いものを基準に他のプログラムのバージョンを揃える．例えば VM_Send.py, または Module.py が ver.2.0 の場合, operational_requirement の要件に適しているため, これに合わせる．Sensor.py が ver.2.0 の場合 latest の要件に適しているため, これに合わせる．更新の初めに切断され, 全てのバージョンが ver.1.0 の場合, required の要件をを満たしているため, 通常動作に移行する．

図 4 では, サーバーでクライアントとのバージョンを比較する際, 例えばクライアント A から受信した情報では, VM_Send.py が ver.2.0, Sensor.py が ver.1.0 であるため, バージョンが一番高い VM_Send.py の ver.2.0 を基準に他のプログラムのバージョンを合わせる．図 5 から推奨される稼働要件の operational_requirement が一番近いバージョンの組み合わせをしている．そのため, operational_requirement を基準としてバージョンを合わせて運用する．図 4 では Module.py のみ, バージョンが稼働条件を満たしていないため, このファイルに対して更新を行う．また, Sensor.py のバージョンも ver.1.0 であるが, 稼働条件を満たしているため更新対象にはならない．その後, 更新が終わり次第, 通常動作に移行し障害復旧とする．

また, サーバーに送られたバージョン情報とサーバーが保持する最低稼働条件のバージョンを比較する際に, 全てのバージョンが稼働条件以上, または最新の状態であった場合, バージョンは稼働条件を満たしている, または更新は全て適用されていると判定し, 追加の更新は行わずデバイスは通常の動作に移行する．

ユースケース・シナリオ

本提案は, 産業用機械のモニタリング機器のアップデートを適用する際を想定している．図 6 にユースケースを示す．産業用機器のモニタリングのため, IoT デバイスを使用している．機能の追加やバグ修正のためにアップデートを行うが, 停電障害により AC アダプターや PoE から直接電力を供給している IoT デバイスのインターネット通信が

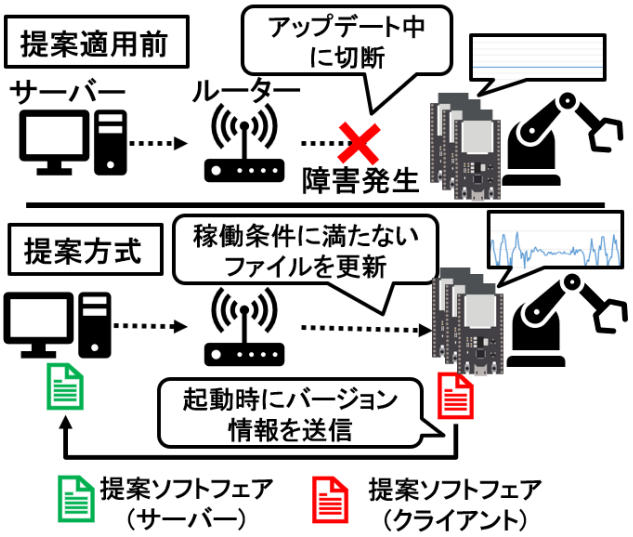

図 6: ユースケース

切断されることを想定する．図 6 の提案適用前ではアップデート中に障害が発生し, IoT デバイスが部分的に更新される．センサーデータが正常に測れなくなりエラーを抱えたままの運用に陥る．本提案を適用することにより, 障害復旧時にバージョン情報を送信し, 更新されていないプログラムを検知して再送信する．これにより, 起動と同時に障害からの復旧が可能となる．

4. 実装

サーバーとクライアントの実装について説明する．また本稿の構成を図 7 に表す．

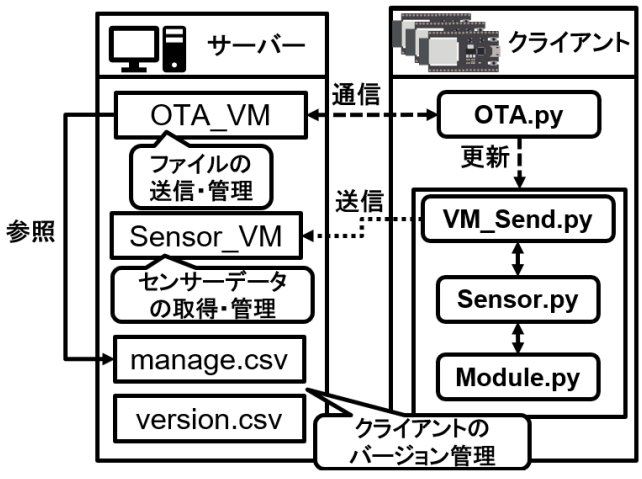

図 7: 構成図

サーバーではクライアントとの通信を行い, アップデートやファイル管理を行う Virtual Machine(以下 VM)『OTA_VM』とセンサーの情報を管理する VM『Sensor_VM』の 2 つサーバーを Ubuntu と Python を用いて実装・開設した．また, クライアントのアップデートファイルの管理を行うための manage.csv とバージョン情報を保存するため

の version.csv を作製した. 提案ソフトウェアは OTA_VM で動作し, クライアントから受信したバージョン情報を version.txt から参照したバージョン管理情報をファイル名同士でバージョンを比較し, 管理情報の稼働条件に満たしていないファイルの名前をクライアントに送信する.

クライアントでは, ESP32 を IoT デバイスとし, 4 つのプログラムを MicroPython を用いて実装した. OTA.py はサーバーとの通信や更新, その他プログラムを動かすための基幹プログラムである. VM_Send.py は取得したセンサーデータを VM 上で開設したサーバーに送信するためのプログラムである. Sensor.py は Module.py を参照しセンサーデータを取得するためのプログラムである. Module.py はセンサーの初期化とデータの処理を行うプログラムである. 提案ソフトウェアは OTA.py で動作し, VM_Send.py, Sensor.py, Module.py 更新対象のプログラム内からバージョン情報取得し, ファイル名とバージョン情報の dict 型のリストを作製し, サーバーの OTA_VM に送信する. バージョンが異なるファイルがある場合, 稼働条件に満たない対象のファイル名を受信し, 再送信が必要なプログラムファイルの更新を行う.

アップデートはクライアントが OTA.py からサーバーの OTA_VM に対してリクエストを送信し, 更新ファイルを受信する. 受信したファイルを更新対象である VM_Send.py, Sensor.py, Module.py に更新を行う. 更新完了後, VM_Send.py は Module.py を参照し, Sensor.py で取得したセンサーデータをサーバーの Sensor_VM に送信する.

5. 評価実験

評価実験として, アップデート中の切断を行い, ファイルの欠落が 1 つ, 2 つ, 全ての場合の複数のケースで更新されていないファイルの検出率によって未更新のファイルが正しく判定されているかを評価する. 次に, それぞれのケースで初めからアップデートを行った時間とバージョンが異なるファイルすべてを再送信した時間, 提案を用いて送信するファイルを管理して再送信した時間を比較し, 再送信と復旧に要した時間が削減されたかの評価を行う.

実験環境

実験は以下の環境で実施する.

クライアント

- ESP32
  Wi-Fi および Bluetooth 機能を備えた低コストで低消費電力のマイクロコントローラである [19]. 本稿では IoT デバイス (クライアント) として使用した.
- BMP280
  Bosch Sensortec 社が開発した環境センサーで, 気圧と温度を測定するために設計されている. BMP280 センサーは, 空気圧の測定に広く使用されている [20]. 本稿ではアップデート前後でセンサー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組み込みシステム向けの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)
  クライアントとの通信, 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, Ubuntu24.04LTS をインストールした仮想マシンを用意した.
- Python SimpleHTTPRequestHandler
  ESP32 デバイスに遠隔更新を行うためにするために使用した. 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.

基礎実験

基礎実験では, クライアントがアップデートを完了した状態と, アップデート中に通信が切断された状態で, サーバーに送信されるセンサーデータへの影響を調査した. アップデートは提案と同じ条件で行う. クライアントの温度データのみを取得する既存プログラムに, 気圧データを取得する機能を追加するために更新を行った. 通信を切断した場合の更新されなかったプログラムは Sensor.py と Module.py の 2 つであると想定する. アップデートが完了した状態のサーバーのターミナルの出力結果を実行結果 1 に示す.

実行結果 1: アップデート完了時のサーバーログ

```
1  Starting HTTP server on
   192.168.100.137:8000...
2  Received data: Temperature = 28.19 ° C,
   Pressure = 931.3025 hPa
3  192.168.100.94 - - [29/Jul/2024 06:25:11]
   "POST / HTTP/1.0" 200 -
4  Received data: Temperature = 28.19 ° C,
   Pressure = 931.3323 hPa
5  192.168.100.94 - - [29/Jul/2024 06:25:16]
   "POST / HTTP/1.0" 200 -
6  Received data: Temperature = 28.2 ° C,
   Pressure = 930.7073 hPa
7  192.168.100.94 - - [29/Jul/2024 06:25:21]
   "POST / HTTP/1.0" 200 -
8  Received data: Temperature = 28.2 ° C,
   Pressure = 930.8261 hPa
9  192.168.100.94 - - [29/Jul/2024 06:25:26]
   "POST / HTTP/1.0" 200 -
10 ・・・
```

1 行目は, サーバーを起動した状態である. 2, 4, 6, 8 行目はクライアントから受信した温度 (Temperature) と気圧 (Pressure) のセンサーデータを表示している. 実行結果 1 から, 温度と気圧データが受信できている事から, アップデートが完了した状態であるとわかる. 3, 5, 7, 9 行目はクライアントからサーバーにセンサーデータを送信した際の HTTP サーバーのログメッセージである. このメッセージは HTTP ステータスコードが 200 であるため, 通信におけるエラーは無くリクエストは成功している.

次にアップデート中に通信が切断された状態のターミナルの出力結果を実行結果 2 に示す.

実行結果 2: アップデート中断時のサーバーログ

```
1  Starting HTTP server on
   192.168.100.137:8000...
2  Received data: Temperature = 28.19 ° C,
3  192.168.100.94 - - [29/Jul/2024
   06:21:32]"POST/HTTP/1.0"200 -
4  Received data: Temperature = 28.2 ° C,
5  192.168.100.94 - - [29/Jul/2024 06:21:37]
   "POST / HTTP/1.0" 200 -
6  Received data: Temperature = 28.2 ° C,
7  192.168.100.94 - - [29/Jul/2024 06:21:42]
   "POST / HTTP/1.0" 200 -
8  Received data: Temperature = 28.21 ° C,
9  192.168.100.94 - - [29/Jul/2024 06:21:47]
   "POST / HTTP/1.0" 200 -
10 ・・・
```

1 行目及び 3, 5, 7, 9 行目は実行結果 1 と同様の動作であるため説明は省略する. 2, 4, 6, 8 行目を見ると, クライアントから送信されるデータの型には温度 (Temperature) と気圧 (Pressure) の 2 つが確認できる. 実行結果 2 から, 温度データには数値があるため受信できている事がわかるが, 気圧データが出力されていない. クライアントは気圧データを取得する機能の追加ができていない. そのため, アップデートの中断によりプログラムの更新がされず, 気圧データの取得と送信がされていない.

6. 議論

本稿では, 起動時にクライアントがサーバーに接続する際, バージョン情報のみを送信し, サーバーが保持するバージョン管理ファイルと比較して稼働条件を満たしているか確認を行う. サーバー側でクライアントに送信するファイルを管理することを目的としている.

バージョン管理ファイルではプログラムごとに稼働条件としてバージョンを指定する管理方法を用いたため, プログラムごとの依存関係を考慮していないことになる. 考慮しないことで全てのバージョンを比較しなければならない. そのため, バージョン管理ファイルでバージョンごとに依存関係のバージョンの範囲を指定することで解決できる.

7. おわりに

課題は IoT デバイスを遠隔でアップデートする際, 通信が切断されると更新ファイルが完全には適用されず, 一部のファイルが欠落した状態で更新に陥ってしまう事である. 直接 AC アダプターや PoE から電力を供給している IoT デバイスは停電障害により接続がせつだんされてしまう. 提案ではデバイスを起動した時, 又は障害復旧時にサーバーに接続しデバイスが保持しているバージョン情報を送信する. サーバーは送られてきたバージョン情報と保持しているバージョン管理ファイルのバージョン情報を比較し, 稼働条件に満たないプログラムのみをデバイスに送信する. これにより, 欠落したプログラムファイルのバージョンを稼働条件に合わせることで, 迅速な障害復旧が可能となる.

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] 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).

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

[6] Zikria, Y. B., Kim, S. W., Hahm, O., Afzal, M. K. and Aalsalem, M. Y.: Internet of Things (IoT) operating systems management: Opportunities, challenges, and solution, *Sensors*, Vol. 19, No. 8, p. 1793 (2019).

[7] Wang, Q., Zhu, X., Ni, Y., Gu, L. and Zhu, H.: Blockchain for the IoT and industrial IoT: A review, *Internet of Things*, Vol. 10, p. 100081 (2020).

[8] Valach, A. and Macko, D.: Exploration of the LoRa technology utilization possibilities in healthcare IoT devices, *2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, IEEE, pp. 623–628 (2018).

[9] 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).

[10] Vaniea, K. and Rashidi, Y.: Tales of software updates:

The process of updating software, *Proceedings of the 2016 chi conference on human factors in computing systems*, pp. 3215–3226 (2016).

[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] 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 (2019).

[13] Yohan, A. and Lo, N.-W.: An over-the-blockchain firmware update framework for IoT devices, *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, IEEE, pp. 1–8 (2018).

[14] 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).

[15] Guo, Z., McDirmid, S., Yang, M., Zhuang, L., Zhang, P., Luo, Y., Bergan, T., Musuvathi, M., Zhang, Z. and Zhou, L.: Failure recovery: When the cure is worse than the disease, *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)* (2013).

[16] Kuźniar, M., Perešíni, P., Vasić, N., Canini, M. and Kostić, D.: Automatic failure recovery for software-defined networks, *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 159–160 (2013).

[17] Hu, Y.-L., Cho, Y.-Y., Su, W.-B., Wei, D. S., Huang, Y., Chen, J.-L., Chen, I.-Y. and Kuo, S.-Y.: A programming framework for implementing fault-tolerant mechanism in iot applications, *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*, Springer, pp. 771–784 (2015).

[18] Al-Shammari, H. Q., Lawey, A. Q., El-Gorashi, T. E. and Elmirghani, J. M.: Resilient service embedding in IoT networks, *IEEE Access*, Vol. 8, pp. 123571–123584 (2020).

[19] 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).

[20] 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).

