卒業論文

リアルタイム小型ロケット軌道表示システムの開発と評価

2025 年 3 月

鹿児島大学工学部先進工学科電気電子工学プログラム

指導教員　福島誠治　教授

東迎　健太郎

概要

鹿児島大学では小型ハイブリッドロケットの打ち上げ実験を行なっており，福島研はロケットに搭載する模擬人工衛星を作製してきた．過去の衛星ミッションはテレメトリの無線送受信やアンテナの改良などであった．送受信するテレメトリについて，過去の衛星では加速度のみであったが，今回作製する衛星 4 号機では位置情報を含む数種類に増加した．その位置情報を使用し，衛星 4 号機のミッションの一つとして新たに小型ロケットのリアルタイム軌道表示を行う．本研究ではリアルタイム軌道表示を実現するためのシステムの開発および評価を行なった．システムの構成要素は衛生局，地上局, Web サーバー, Web ブラウザである．本文はこれらの技術構成，実装ロジック，実験，評価という章立てである．

目次

第 1 章　序論

鹿児島大学では 2020 年度から毎年度 1 回ずつ小型ハイブリッドロケットの打ち上げ実験を行ってきた．この打ち上げ実験の参加者は，鹿児島大学から工学部先進工学科電気電子工学 PG の福島研と工学部先進工学科機械工学 PG の片野田研が，外部からは鹿児島ハイブリッドロケット研究会（KROX）やスポンサー企業が参加している．

ハイブリッドロケットは，「燃料に固体（プラスチックなど），酸化剤に液体（液体酸素など）を用いる場合が多い．ハイブリッドロケットは燃料が爆発する心配がなく，また扱う液体が 1 種類であるため，液体ロケットや固体ロケットに比べて安全で小型化が可能であり，そのため安価に製造できるという特徴がある [1)]」と説明されている．片野田研はこのハイブリッドロケット本体を，福島研はハイブリッドロケットに搭載する小型模擬人工衛星を作製してきた．

表 1 に過去の打ち上げ実験の実績を示す．人工衛星を初めてハイブリッドロケットに搭載したのは 2022 年 3 月に打ち上げたロケット 3 号機である．なお，ロケット 6 号機（人工衛星 4 号機）の打ち上げ時期は本論文の執筆時点（2024.2）では定まっていない．

表 1　打ち上げ実験の実績

ロケットの機体番号	初号機	2 号機	3 号機	4 号機	5 号機
人工衛星の機体番号			初号機	2 号機	3 号機
打ち上げ年月	2019.12	2020.12	2022.3	2023.3	2024.2

第 2 章　目的

2025 年度の打ち上げ実験で使用する人工衛星 4 号機では新たにリアルタイムロケット軌道表示 (RTTD) というミッションを追加する．本論文ではこの RTTD システムの設計から開発・評価までを記す．

RTTD ミッションで実現することは以下の内容である．

・ロケットがある地点で取得した位置情報をリアルタイムでユーザーに知らせる．

・打ち上げ実験場にいない人にも打ち上げの様子を提供する．

上記 2 点を実現するために，RTTD システムはインターネット環境にいるユーザーを対象とし，地図上にロケットの位置情報をリアルタイム表示するようなシステムにする．ユーザーは図 1 の右側のスマートフォンに示されるような画面でロケットの位置を確認することができる．

図 1　RTTD ミッション完成像

第 3 章　使用技術

3.1 ~ 3.5 に RTTD で使用する主な技術または機器とその役割を示す.

3.1 衛星局の使用技術

(1) Raspberry Pi 4（RP4）

シングルボードコンピュータ. OS（本研究では Raspberry Pi OS）を載せることができるため，衛星局内でプログラムを動かすことができる.

(2) Python

インタプリタ型言語. 無線モジュールを操作するための RPi.GPIO や GPS センサを操作するための micropyGPS といった便利な Python 用ライブラリを使用することができる.

(3) IM920sL

特定小電力無線である 920MHz 帯の無線送受信を行う.

(4) Grove 社 GPS モジュール

位置情報センサ. パラメータは緯度・経度（WGS84），高度，速度，タイムスタンプの 5 種類.

(5) BME280

温湿度・気圧センサ. パラメータは温度，湿度，気圧，高度の 4 種類.

(6) MPU-6050

6 軸ジャイロセンサ. パラメータは加速度（x 軸, y 軸, z 軸），角速度（x 軸, y 軸, z 軸）の 6 種類.

(7) USB マイク

小型モノラルマイク. ロケットの打ち上げ時や上空で音声の録音を行う.

(8) スパイカメラモジュール

HD ビデオカメラ. RP4 との接続部からカメラまで 300 mm あるため，カメラを衛星の外に出してロケット表面に穴を開けることで外部の映像を撮影することができる.

(9) 圧電スピーカー

RP4 に電源を投入した際に音を鳴らすことで電源投入者へ合図を出すことができる.

3.2 地上局の使用技術

(1) Raspberry Pi 4（RP4）

前述. 地上局でプログラムを動かすことができる.

(2) Go

コンパイラ型言語. Google によってシンプルで効率的な言語として開発された. 特に並行処理が強力. Go の並行処理はゴルーチンというスレッドのようなものを使用し, 通常の OS スレッドよりも非常に軽量で数千から数百万単位で並行動作させることが可能. Go はスレッド（ゴルーチン）間のデータ共有にチャネルというものを使用する. チャネルはメッセージパッシングモデルであるため, 開発者が排他制御を行う必要がなく Go ランタイムが自動的にゴルーチンとチャネルのスケジューリングを管理する.

(3) IM920sL

前述.

(4) gRPC クライアント

gRPC は Google が開発した RPC（Remote Procedure Call）フレームワークである. g は gRPC のバージョンによって意味が異なる [2]. RPC とは, ネットワークを介してリモートのコンピュータ上にある関数やプロシージャをローカルの関数を呼び出すかのように実行できる仕組みである. gRPC はデータフォーマットにプロトコルバッファ（Protobuf）を使用するため, スキーマ（.proto ファイル）に記述したデータ構造を元にバイナリ形式にシリアライズしてデータ送信を行う. また, gRPC は 4 つの RPC モデルを提供しており, リクエストとレスポンスが一対一である Unary RPC, 一対多である Server Streaming RPC, 多対一である Client Streaming RPC, 多対多である Bidirectional Streaming RPC がある.

RTTD システムでは地上局からバックエンドサーバーに位置情報を一つずつ送信するため, スキーマにクライアントストリーミングを定義し, gRPC クライアントを実装する必要がある.

(5) モバイルルーター

地上局とバックエンドサーバーの間で通信を行うために，モバイルネットワークを介してインターネットに接続することができる．

3.3 インフラストラクチャーの使用技術

本研究においてインフラストラクチャーはバックエンドのネットワーク構築を指す．これには AWS（Amazon Web Service）というクラウドサービスを使用した．AWS で使用したサービスまたは構成要素は第 4 章 4.3 に記す．

3.4 バックエンド

(1) Go

前述．

(2) gRPC サーバー

前述．クライアントストリーミング RPC を持つ gRPC サーバーを実装する必要がある．

(3) Echo

Go 向けの高速で拡張性の高い軽量 Web フレームワーク．リアルタイムアプリケーション（WebSocket や HTTP/2）のサポートがされている．ブラウザとバックエンドサーバー間の REST API の実装に使用する．

(4) PostgreSQL

RDBMS（Relational Database Management System）の一つ．リレーショナルモデルに基づいてデータを管理するデータベース管理システム．データを表（テーブル）形式で管理し，それぞれの表が列（カラム）と行（レコード）から構成される．データの操作（挿入，更新，削除，検索など）や定義には SQL（Structured Query Language）が使用される．

RTTD システムにおいては大量の位置情報や時間ログの保存に使用する．

(5) GORM

Go の ORM（Object-Relational Mapper）ライブラリ．データベース操作をより簡単かつ直感的に Go で記述することができる．MySQL, PostgreSQL, SQLite, SQL Server といった主要な RDB をサポートしている [2)]．

3.5 フロントエンド

(1) Next.js

React をベースにしたフロントエンドフレームワーク. React とは Facebook(現 Meta) によって開発されたフロントエンドライブラリで, 再利用可能な UI コンポーネントを作成することでアプリケーションを構築し, データ（ステート）が変わると React が自動的に UI を再描画する. Next.js は React に幾つかの機能を加えたものである. レンダリングに関する機能としてはサーバーでページを生成してクライアントに送信するサーバーサイドレンダリング (SSR) や, ビルド時に静的な HTML ファイルを生成したり一度生成した HTML をキャッシュしたりする静的サイト生成（SSG）などがあり, 高速にレンダリングできるようになる. 効率化の機能としてはフォルダ構造に基づいて URL が自動的に定義されるファイルベースのルーティングや, 画像の遅延読み込みやサイズ変更などを自動で最適化する機能などがある.

(2) TypeScript

Microsoft が開発した JavaScript のスーパーセットであり, 静的型付けをサポートしたプログラミング言語. TypeScript のコードはブラウザや Node.js が直接実行できるように JavaScript にトランスパイルされる.

(3) TailwindCSS

ユーティリティファーストの CSS フレームワーク. 小さなユーティリティクラスを組み合わせて UI を構築. HTML（Next.js と React においては仮想 DOM）内でスタイリングを記述するため, CSS ファイルほとんど書かずにデザイン可能.

(4) DeckGL

Uber が開発した地図データや地理空間データを視覚化するための高性能な Web フレームワーク. WebGL を基盤として動作するため, 大規模なデータセットの効率的なレンダリングが可能. Mapbox や Google Maps などの地図ツールと簡単に統合できる.

(5) Vercel

フロントエンドプロジェクトのホスティングやデプロイを簡単に行えるプラットフォーム. Next.js は Vercel によって開発されたため特に Next.js との統合が強力で, Next.js アプリケーションを迅速かつ簡単にデプロイできるように設計されている.

第 4 章　実装

4.1 ~ 4.5 に RTTD のコンポーネントごとのロジックを示す.

4.1 衛星局

衛星局の動作ロジックは(1) ~ (11)の順であり，そのフローチャートは図 2 の通りである.

(1) RP4 の電源投入時に圧電スピーカーを初期化し，音を鳴らす.
(2) IM920sL を初期化する.
(3) Queue インスタンスを作成する.
(4) Grove GPS の初期化および 1 秒ごとに位置情報を取得し, Queue へ送信するという動作を無限ループするスレッドを起動する.
(5) BME280 の初期化および 0.2 秒ごとに気候情報を取得し, Queue へ送信するという動作を無限ループするスレッドを起動する.
(6) MPU6050 の初期化および 0.2 秒ごとに加速度・角速度を取得し, Queue へ送信するという動作を無限ループするスレッドを起動する.
(7) Queue から各センサデータを受信し, IM920sL で無線送信するという動作を無限ループするスレッドを起動する.
(8) スパイカメラおよび一定時間ごとに録画し，ファイル保存するという動作を無限ループするスレッドを起動する.
(9) USB マイクの初期化および一定時間ごとに録音し，ファイル保存するという動作を無限ループするスレッドを起動する.
(10) IM920sL で無線受信があれば，受信コマンドに従って特定の処理をするという動作を無限ループするスレッドを起動する.
(11) メインスレッドは終了しないように待機処理をする.

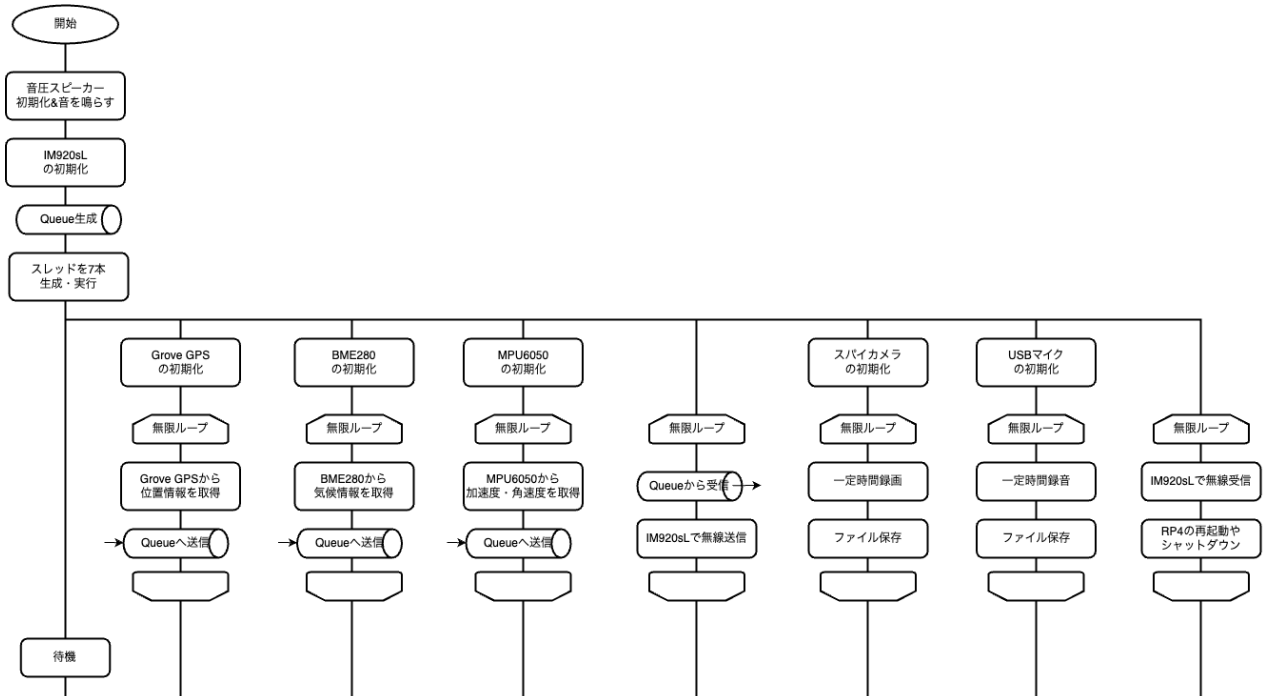


図２　衛星局：動作ロジックのフローチャート

4.2 地上局

地上局の動作ロジックは(1)～(3)の順であり，そのフローチャートは図 3 の通りである．

(1) IM920sL 用シリアルポートを開放および IM920sL 受信のたびにデータ読み取り，ログ出力を行い，ペイロードが位置情報であればチャネルへ送信するという動作を無限ループするゴルーチンを起動する．

(2) gRPC コネクションを確立・クライアントストリームを開始およびチャネルから受信した位置情報を gRPC サーバーへリクエスト送信するという動作を無限ループするゴルーチンを起動する．

(3) メインスレッドは終了しないように待機処理をする．

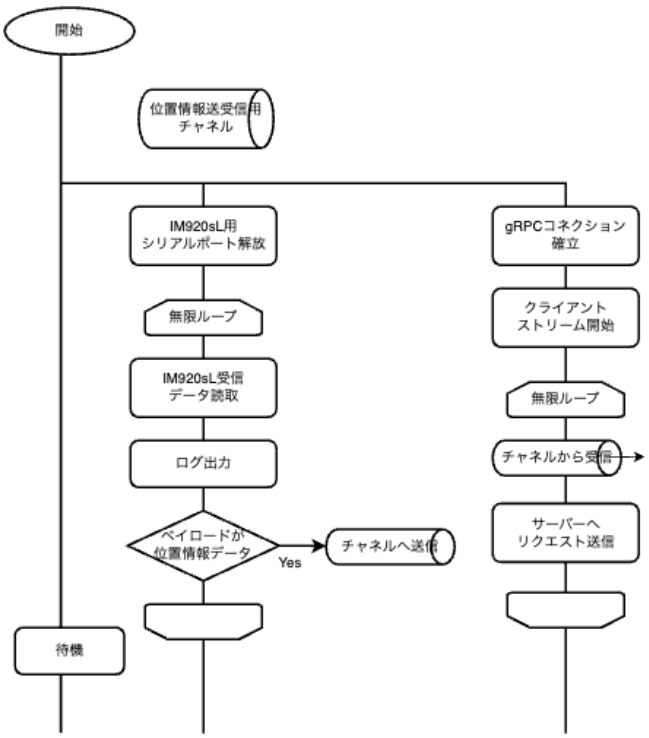


図 3　地上局：動作ロジックのフローチャート

4.3 インフラストラクチャー

インフラストラクチャー構築のために AWS で使用したサービスまたはリソースを以下に示す.これらを用いて構築した AWS 構成図を図 4 に示す.

(1) VPC

データセンターで運用されている従来のネットワークに似た仮想ネットワーク [4].

(2) リージョン

世界各地の地理的に離れた領域 [5]. アジアパシフィック（東京）を使用.

(3) アベイラビリティゾーン

リージョンに複数存在する独立した場所 [5].

(4) サブネット

VPC の IP アドレスの範囲 [4]. パブリックサブネットはインターネットと接続可能で，プライベートサブネットは接続不可能.

(5) インターネットゲートウェイ

VPC とインターネットとの間の通信を可能にする VPC コンポーネント [4].

(6) EC2（Elastic Compute Cloud）

AWS クラウド上の仮想サーバー. 各インスタンスタイプはコンピューティング，メモリ，ネットワーク，ストレージリソースが異なるバランスで構成されている [6]. インスタンスタイプは t2.micro を使用.

(7) RDS（Relational Database Service）

RDS は AWS クラウドでリレーショナルデータベースを簡単にセットアップし，運用し，スケーリングすることのできるウェブサービス [7].

(8) Route53

Route 53 は，可用性が高くスケーラブルなドメインネームシステム（DNS）ウェブサービス [8]. ドメインはお名前.com で取得した [9].

(9) ACM（Amazon Certificate Manager）

ACM はウェブサイトやアプリケーションを保護するパブリックおよびプライベート SSL/TLS X.509 証明書およびキーの作成，保存，更新に伴う複雑さに対処する [10].

(10) ALB（Application Load Balancer）

ロードバランサーは，クライアントにとって単一の通信先として機能します，このロードバランサーは，受信アプリケーショントラフィックを複数のアベイラビリティーゾーンの複数のターゲット(EC2 インスタンスなど）に分散する [11].

(11) EC2 Instance Connect Endpoint および EC2 Instance Connect Endpoint Service

EC2 Instance Connect エンドポイントは ID を認識する TCP プロキシ. EC2 Instance Connect エンドポイントサービスは，IAM エンティティの認証情報を使用して，コンピュータからエンドポイントへのプライベートトンネルを確立する. トラフィックは VPC に到達する前に認証され，承認される [12].

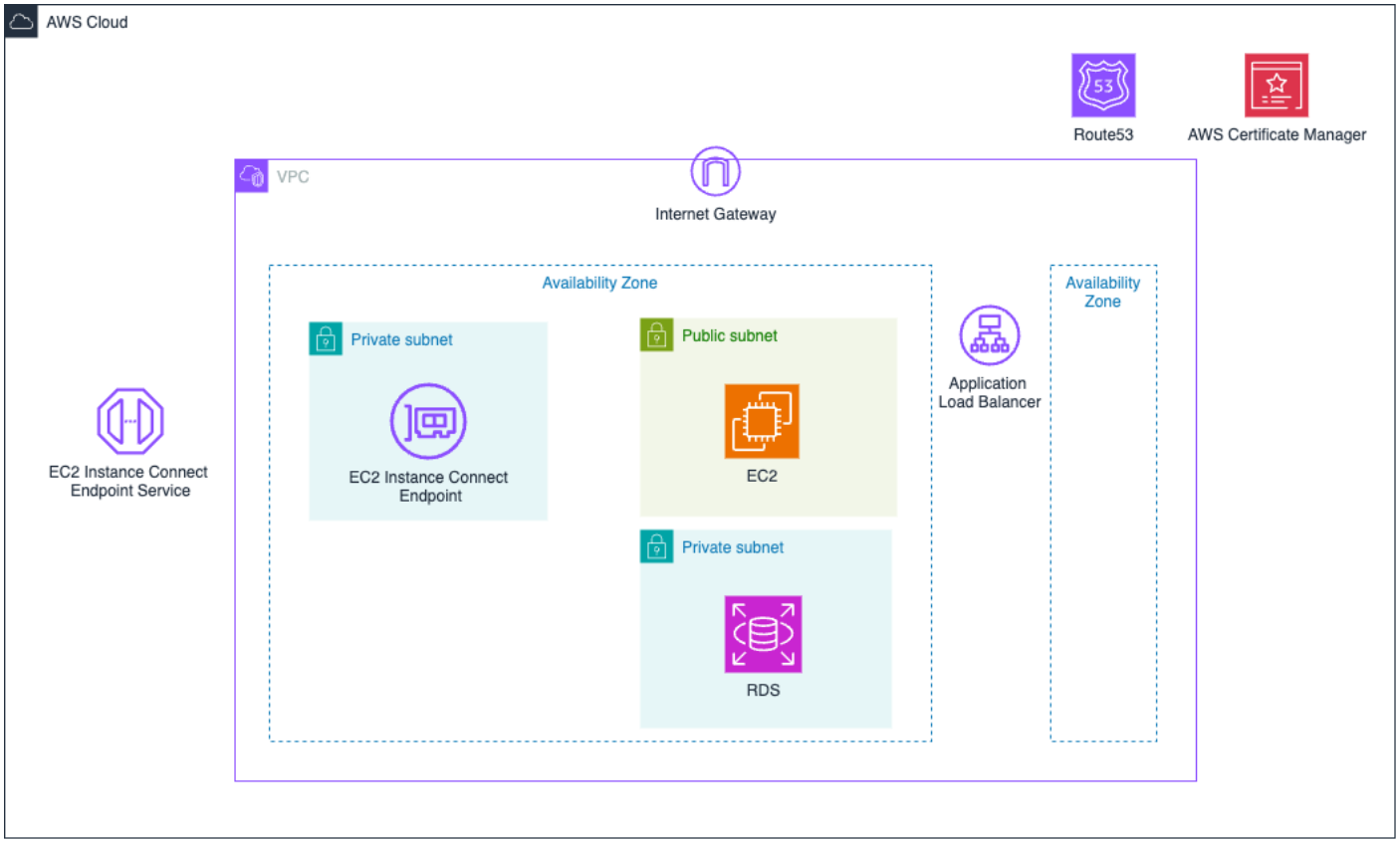


図 4　AWS 構成図

バックエンドサーバーへのシンプルな API のリクエストフローは(1) ~ (4)の順であり，図 5 に示す．

(1) Route53 で DNS 解決を行い，インターネットゲートウェイを経由して ALB の HTTPS エンドポイントに接続する．
(2) ALB は ACM から SSL/TLS 証明書を取得してハンドシェイクを確立する．
(3) ALB から EC2（バックエンド）に HTTP でリクエストを転送する．
(4) EC2 から RDS にアクセスしてデータを取得し，クライアントにレスポンスを返す．

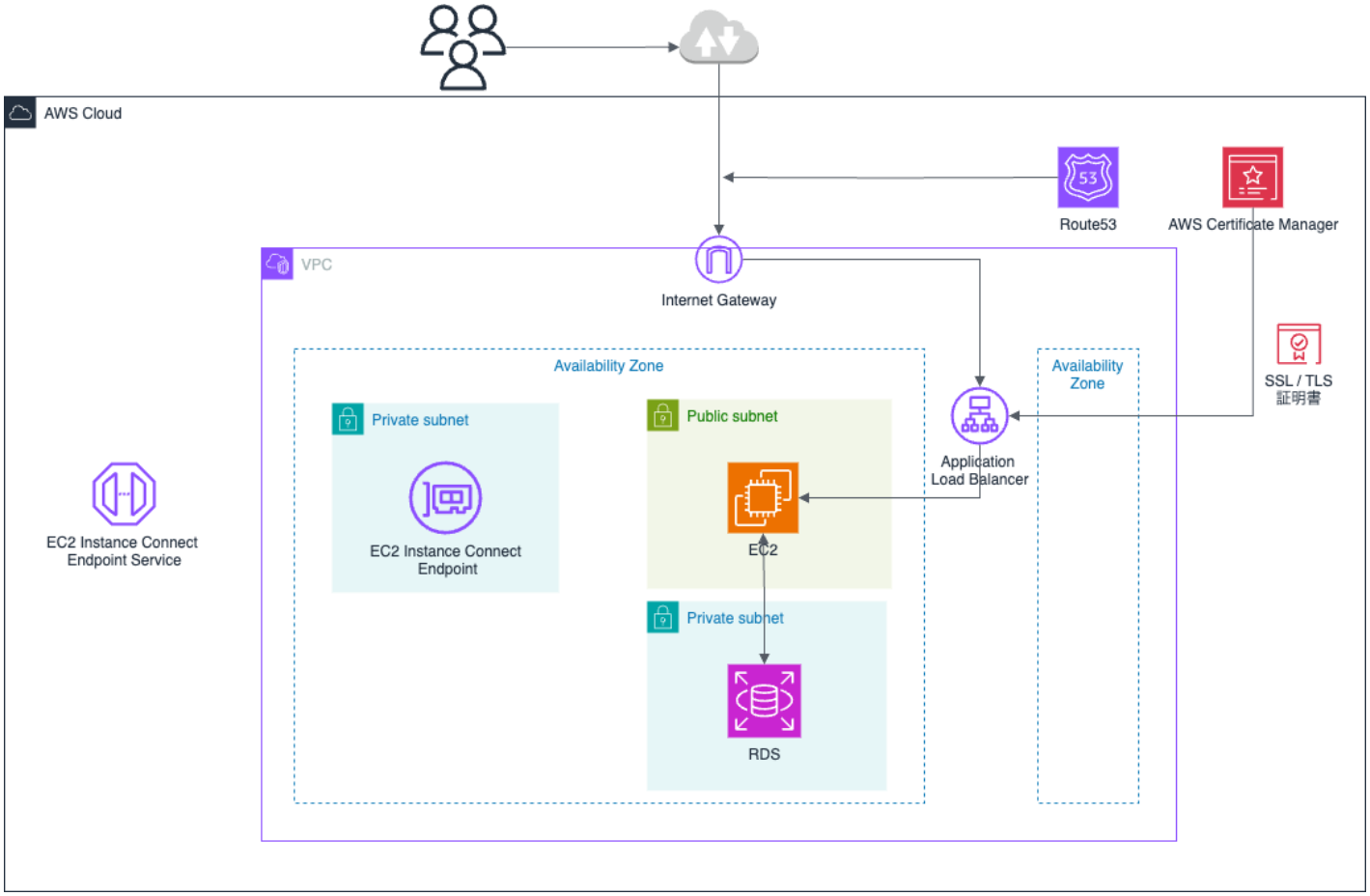


図 5　バックエンドサーバーへのリクエストフロー

また，開発時のインストール作業やデプロイ作業などで EC2 や RDS にセキュアなアクセスをするためのフローを図 6 に示す．事前に EC2 アクセスが可能な IAM ユーザーの設定を行い，AWS CLI コマンドを用いてアクセスする．

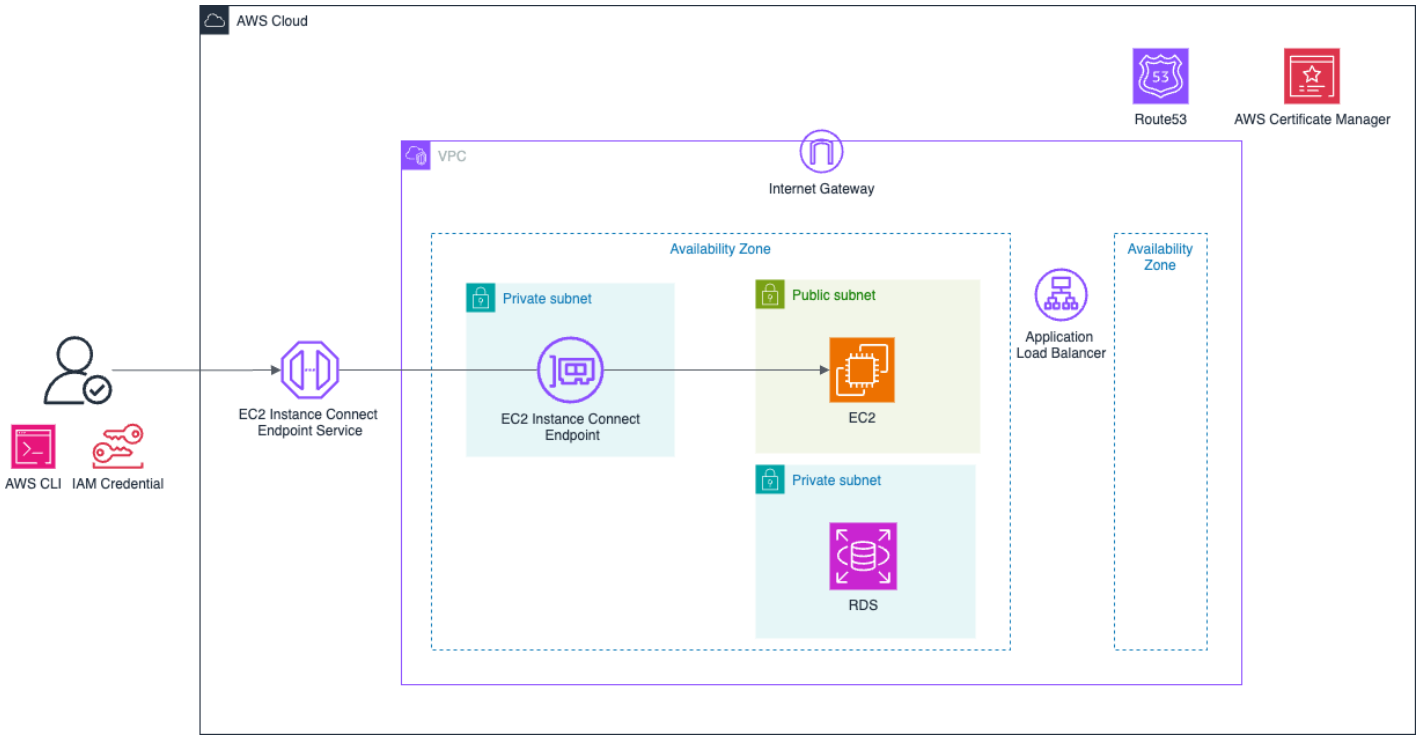


図 6　EC2 への SSH アクセスフロー

4.4 バックエンド

バックエンドの動作ロジックはサーバー起動時, gRPC リクエスト受信時, REST リクエスト受信時の 3 つに分けられる. バックエンドサーバー起動時のロジックは(1) ~ (7)の順であり, そのフローチャートを図 7 に示す.

(1) 2 つのゴルーチンで REST サーバーの起動と gRPC サーバーの起動を開始する.
(2) gRPC サーバー用に特定のポートを開放し, TCP のリスナーを起動する.
(3) gRPC サーバーを初期化し, プロトファイルに定義したテレメトリサービスを埋め込む.
(4) ポートを gRPC サーバーに割り当て, gRPC リクエストを待機する.
(5) Echo エンジンの初期化を行い, API エンドポイントとそれに対応するハンドラを定義する.
(6) REST サーバー用に特定のポートを開放し, TCP リスナーを起動する.
(7) ポートを Echo エンジンに割り当て, REST リクエストを待機する.

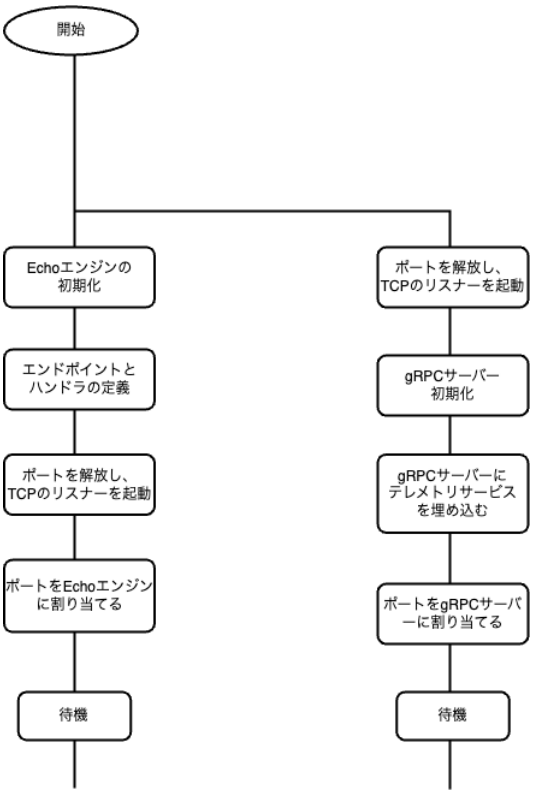


図 7　バックエンド：起動時ロジック

gRPC リクエスト, REST リクエスト処理ロジックはそれぞれ以下の順であり,これらのフローチャートを図 8 に示す.

・gRPC リクエスト

(1) リクエスト（位置情報メッセージ）を受信し DB に保存する.
(2) タイマーが存在しなければ作成・開始する.
(3) 位置情報をキャッシュするため配列に格納する.
(4) クライアントごとに作成された全チャネルに位置情報を送信する.
(5) タイマーをリセットし, 開始する.

・REST リクエスト

(1) GET リクエストを受信し, 本ゴルーチン用のチャネルを作成する.

(2) 位置情報のキャッシュがある場合は全てチャネルに送信され，クライアントにチャンクレスポンスを返す.

(3) チャネルに値が存在しない場合は処理をブロックする.

(4) チャネルから値（位置情報）を取得した場合はクライアントにチャンクレスポンスを返す.

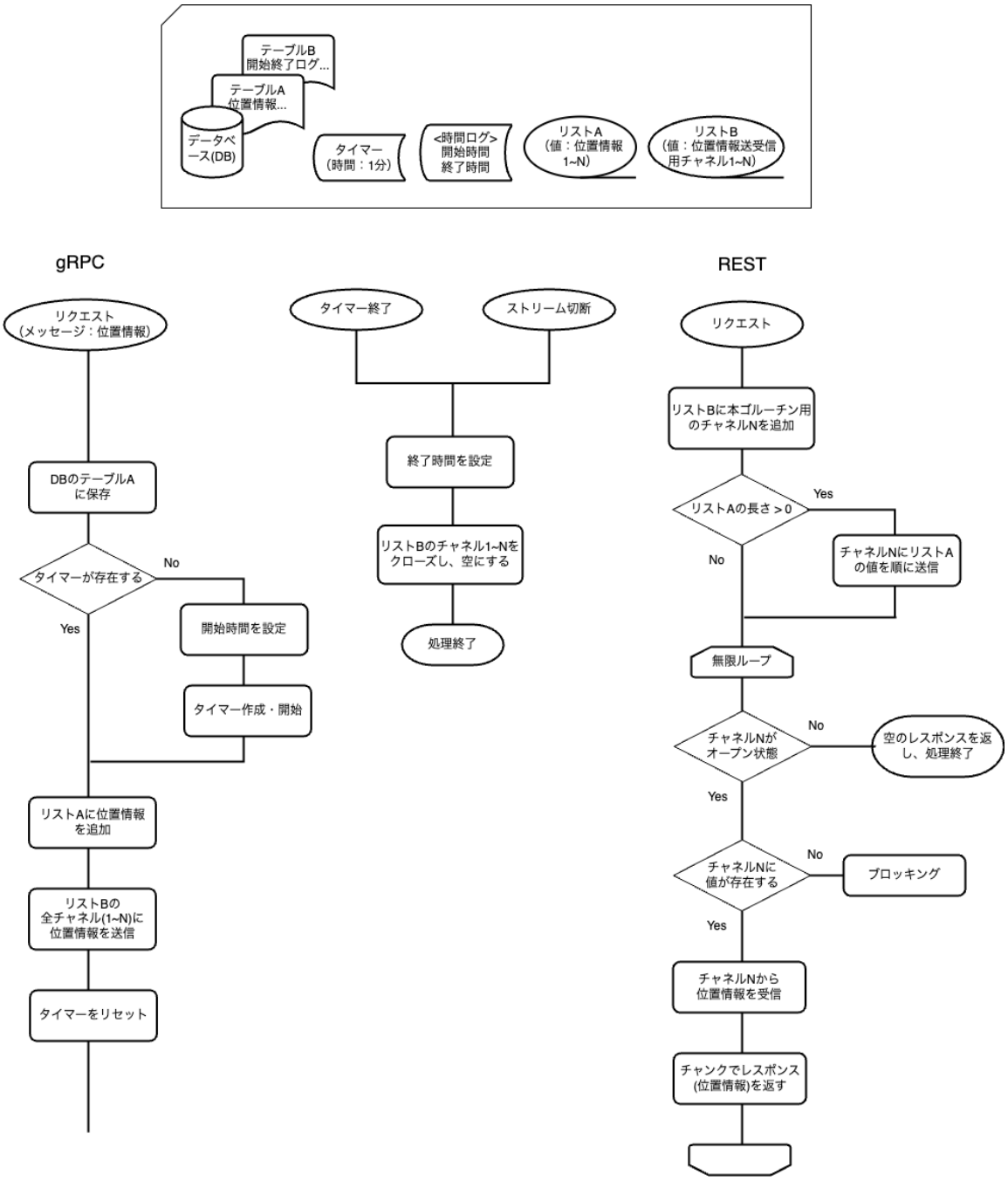


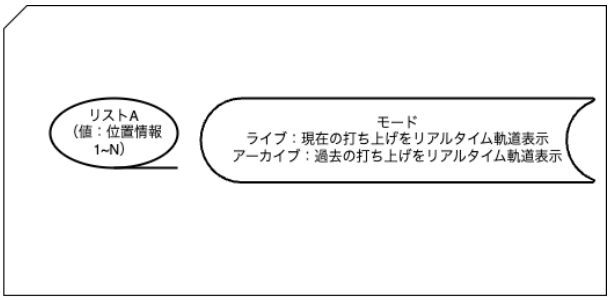
図 8　バックエンド：リクエスト処理ロジック

4.5 フロントエンド

フロントエンドの動作ロジックは UI 操作とブラウザ処理を対応付けて記す．このフローチャートを図 9 に示す．

(1) マップページの URL を入力すると，フロントエンドサーバーからマップページ用ファイルを取得し，UI として表示される．
(2) ライブボタンを押すとモード（ステート）をライブに設定し，バックエンドサーバーのライブ用 API を叩く．その API レスポンスからリーダーを取得する．
(3) 操作をせずに待っていると，リーダーがレスポンス（チャンク）を読み取りリスト A（ステート）に位置情報として追加する．リスト A が変更されると自動的にレンダリングが行われ，マップに新たなプロット点が表示される．

ライブ中にアーカイブボタンを押す，または地上局とバックエンドの接続が切断されるとリスト A を空にし，デフォルトのマップページ画面に戻る．



ステート（State）：変更されるとUIコンポーネントが再レンダリングされる。

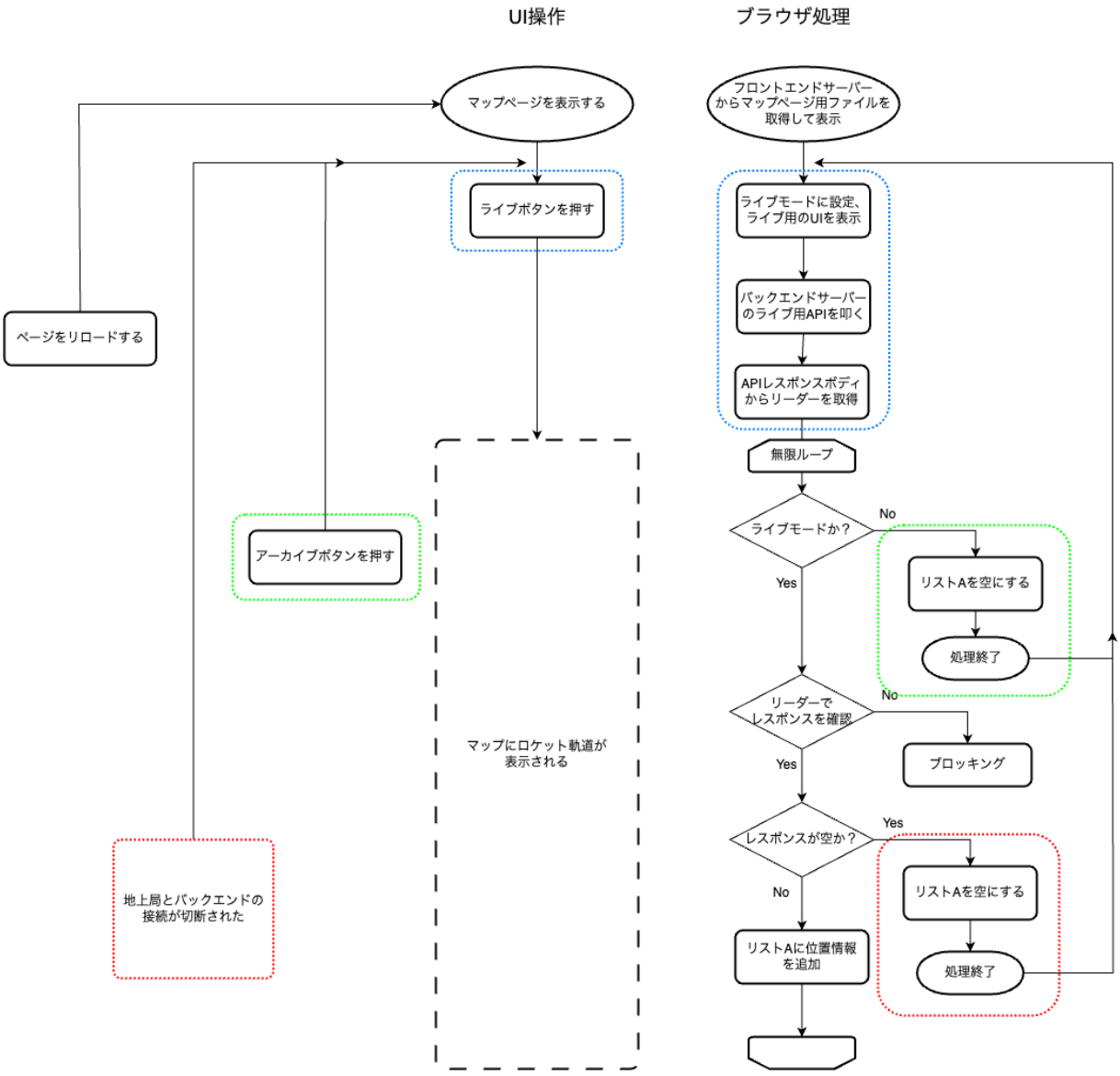


図９　フロントエンド：動作ロジックのフローチャート

第 5 章　実験

5.1 リアルタイム性測定実験

屋外で実際に RTTD システムを動かし，位置情報表示のフロー中にタイムスタンプを記録する．そして，各区間の処理時間やシステム全体のレイテンシおよびジッターを計測する．図 10 に位置情報表示フローおよびタイムスタンプを設ける箇所を示す．

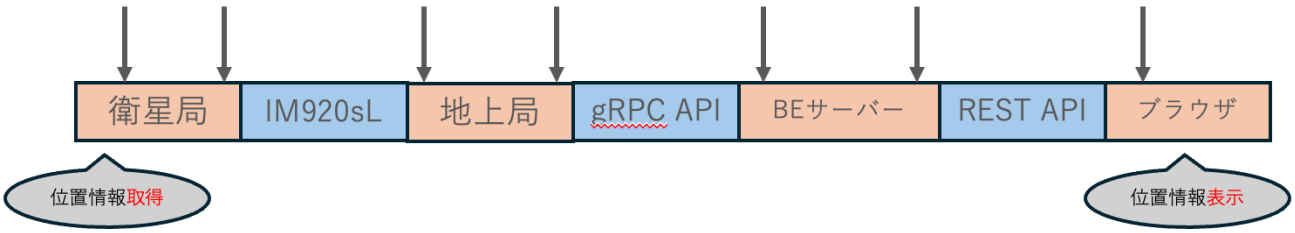


図 10　位置情報取得から表示までのフローおよびタイムスタンプ箇所(矢印)

なおタイムスタンプの記録は衛星局・地上局・BE サーバーにおいてはロガーによるログファイル（.log）への書き込みで行う．ブラウザ上では localStorage への書き込みで行う．localStorage 以外にも sessionStorage や indexedDB などがあるが，データ永続化や書き込み速度の観点から localStorage を選択した．

実験の条件を以下に示す．

・鹿児島大学農学部農場で行う．

・地上局と衛星局の距離を 10 m 程とる．（RP4 はインターネットと接続することで正確な時間を刻めるため，衛星局もモバイルルーターに接続できる範囲にいる必要がある．）

・マップ上に 100 点ほどプロットを行う．これを 2 回繰り返す．

リアルタイム性測定実験の結果を以下に示す.

まず, 位置情報表示フローにおける各区間での処理時間を表 2 に示す.

表 2　各区間での処理時間

試行[回目]	データ数[個]	衛星局[ms]		地上局[ms]		バックエンド[ms]		ブラウザ[ms]	全体[ms]
		取得	終了	開始	終了	開始	終了	表示	
1	115	13.17	37.36	1.09	100.36	3.34	58.73		214.06
2	115	14.92	87.56	0.21	65.89	3.53	65.03		237.14
3	126	12.66	93.37	0.26	69.62	3.81	66.65		246.37

試行 3 回の全体の時間の平均より, レイテンシは 232.52 ms である.

次に, ブラウザで記録したタイムスタンプを順に並べ, それぞれの間の時間を計算する. これはブラウザへのデータ到達時間間隔の変動を表し, 標準偏差を計算することでジッターが求められる. 図 11~13 にブラウザへのデータ到達時間間隔の変動を示す.

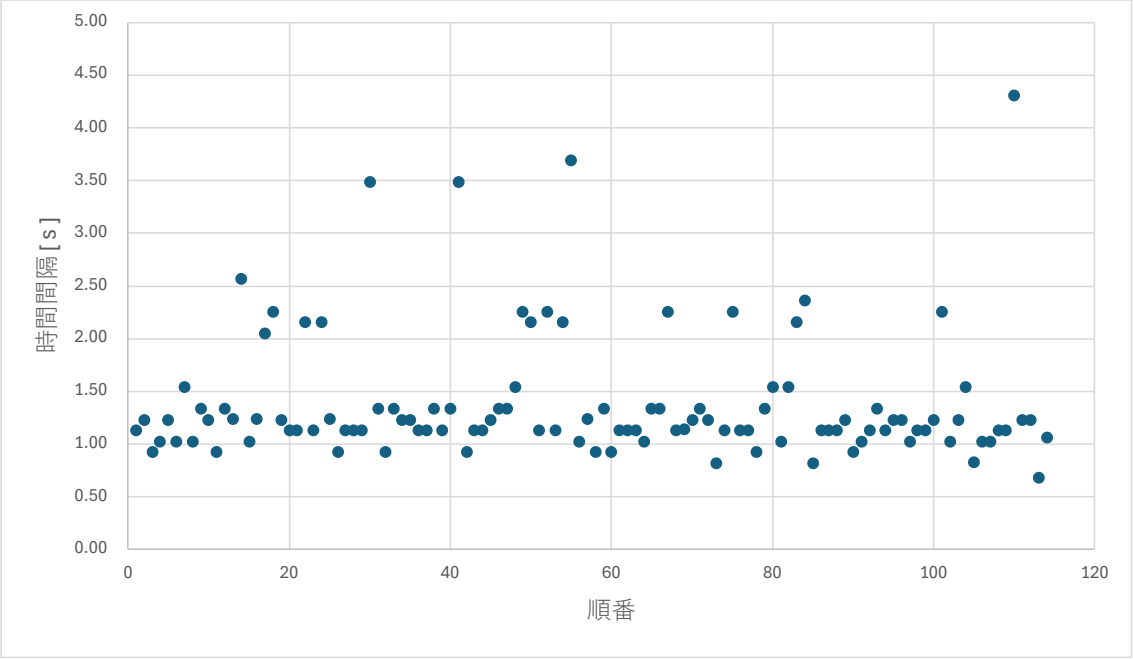


図 11　ブラウザへのデータ到達時間間隔の変動（試行 1）

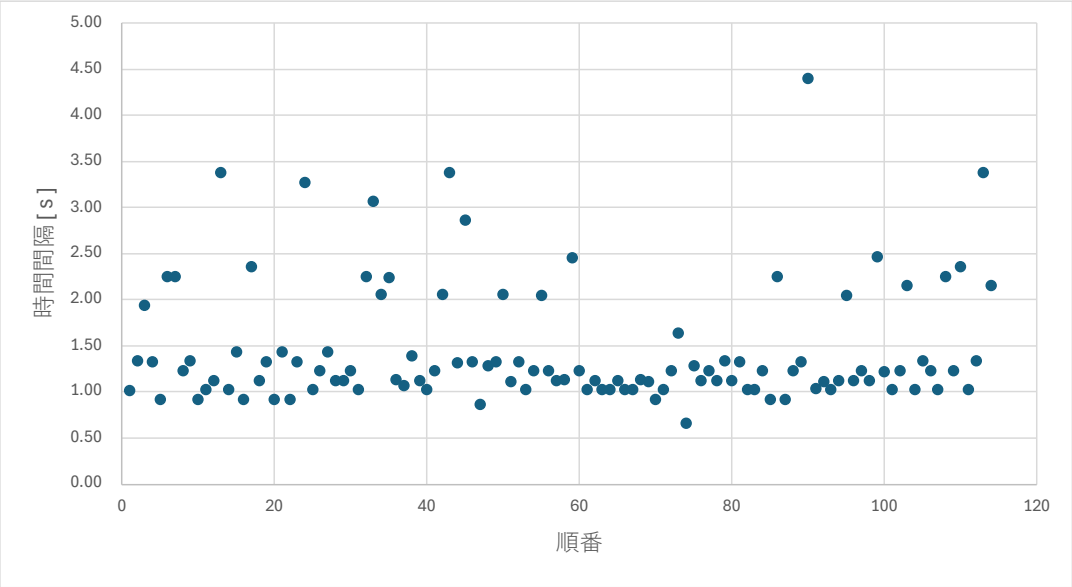


図 12　ブラウザへのデータ到達時間間隔の変動（試行 2）

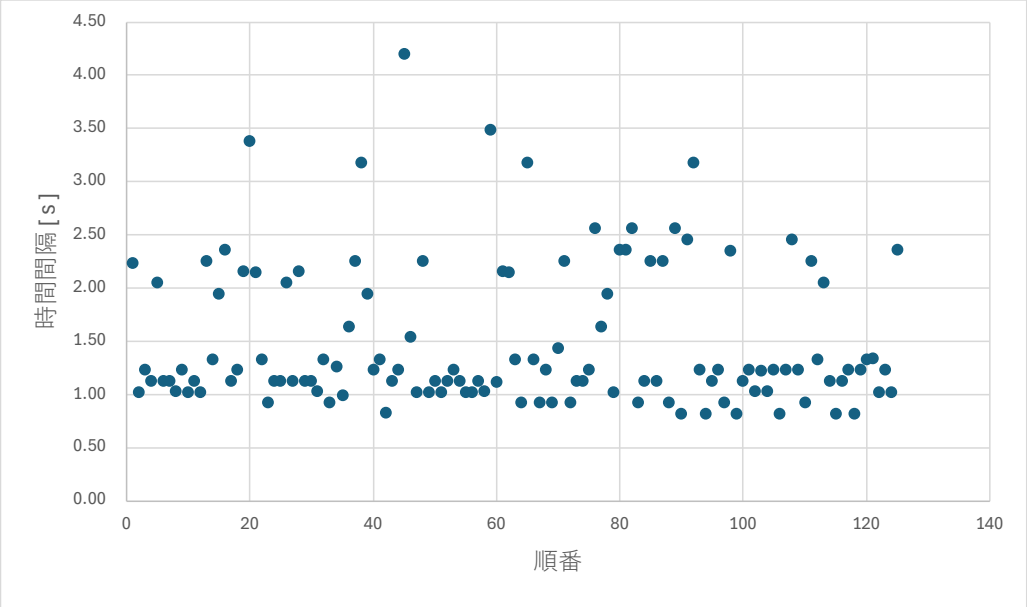


図 13　ブラウザへのデータ到達時間間隔の変動（試行 3）

データ到達時間間隔の平均を求める．また，標準偏差を計算することでジッターも求める．

試行 1　データ到達時間間隔：1.375 秒，ジッター：0.596 秒

試行 2　データ到達時間間隔：1.452 秒，ジッター：0.654 秒

試行 3　データ到達時間間隔：1.493 秒，ジッター：0.667 秒

5.2 負荷耐性測定実験

RTTD システムに接続するクライアント数が増加したとき，システムが各クライアントに対してリアルタイム軌道表示を提供できるかを試す.

実験の条件（図 14）を以下に示す.

・一台の PC から gRPC リクエスト（地上局-バックエンド間）, REST リクエスト（ブラウザ-バックエンド間）を行う.

・gRPC リクエスト（クライアントストリーム）は 100 個の位置情報（仮）を 1.5 秒間隔で送信する.

・クライアント（仮想ユーザー）は 100 ~ 1000 人で行う.

・ネットワークのスループットは 123.1Mbps(UL),72.1Mbps(DL)である.

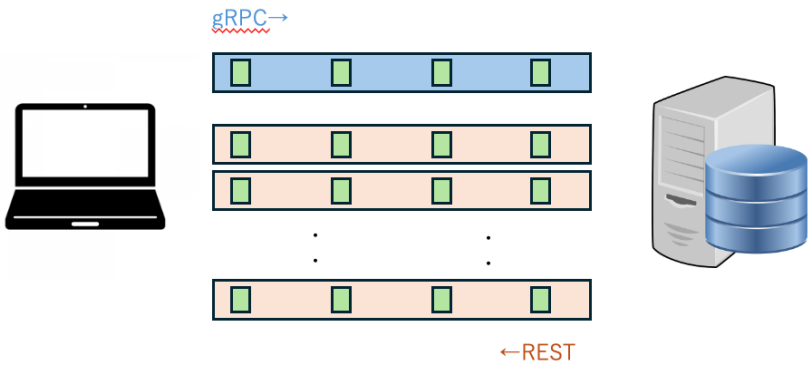


図 14　負荷耐性測定実験概要

負荷耐性測定実験の結果を以下に示す．仮想ユーザーが 100 ~ 1000 人において，全てのデータを取得できた人数と途中でリクエストが切断された人数を図 15 に示す．

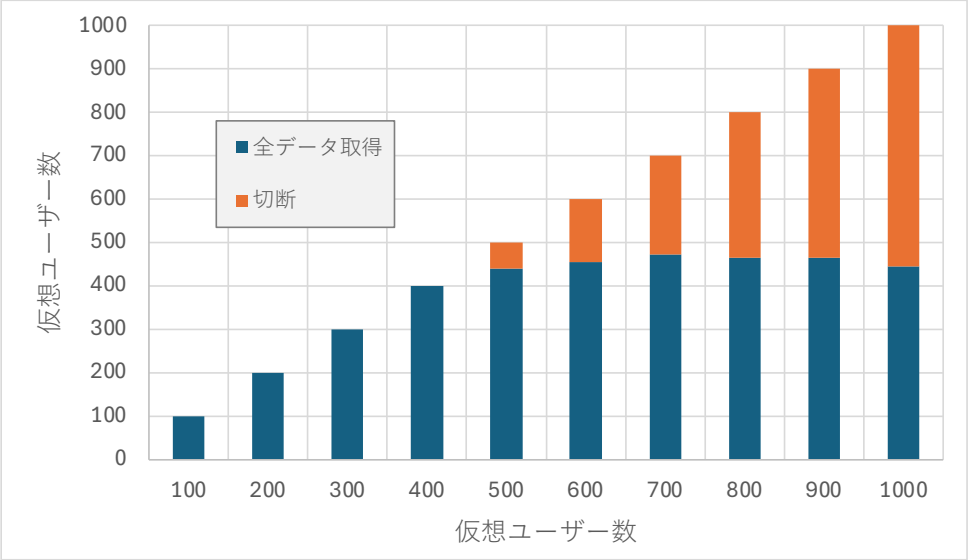


図 15　負荷耐性測定実験結果

図 15 より本実験環境においては RTTD システムに接続可能なクライアントの人数は約 450 人である．

次に，仮想ユーザーが 100人の時の各リクエスト時間を図 16, 100人の中で最も早い開始時間または終了時間を基準とした時の相対時間を図 17 に示す．同様に仮想ユーザーが 300人の時は図 18，図 19 に示す．

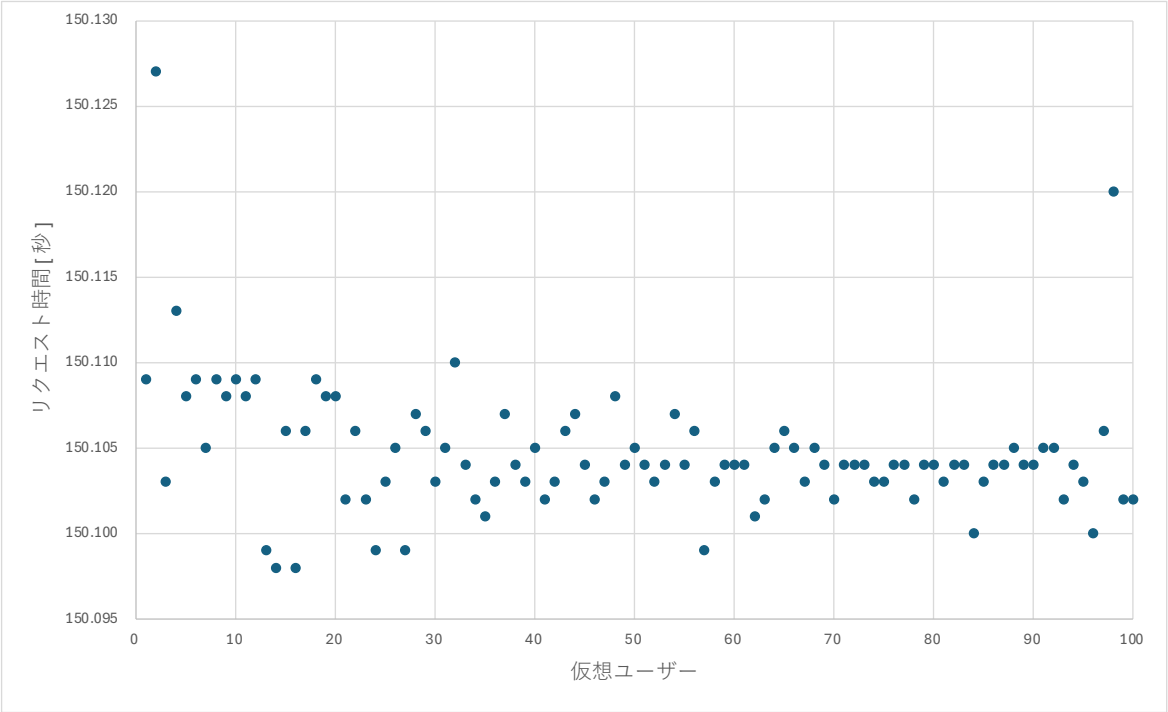


図 16　仮想ユーザー(100 人)のリクエスト時間

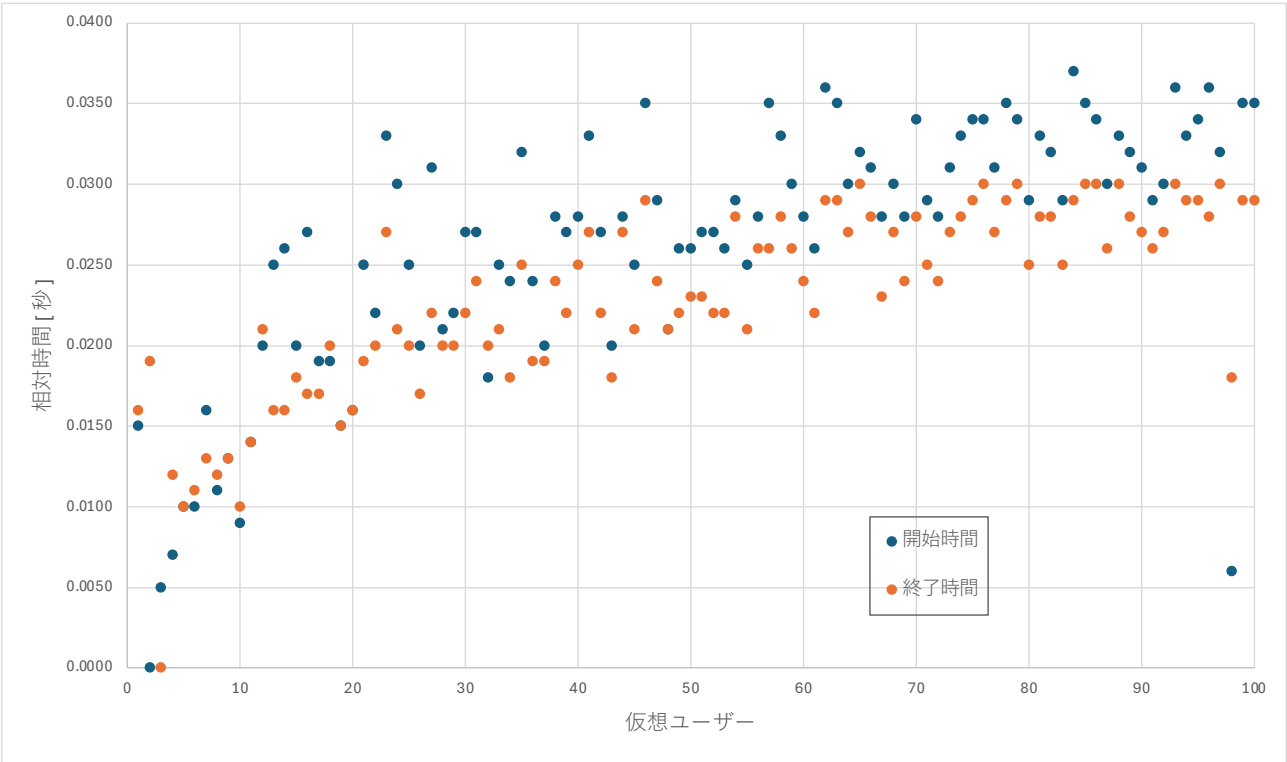


図 17　仮想ユーザー(100 人)のリクエスト開始・終了時間

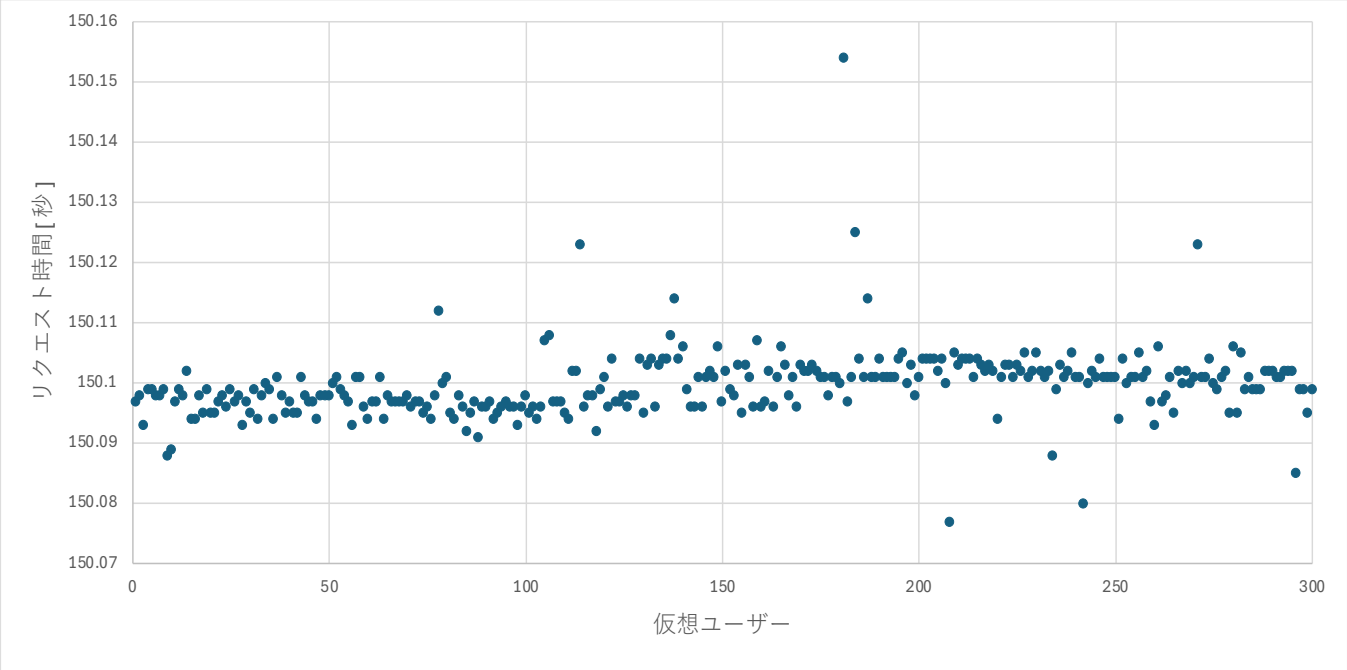


図 18　仮想ユーザー(300 人)のリクエスト時間

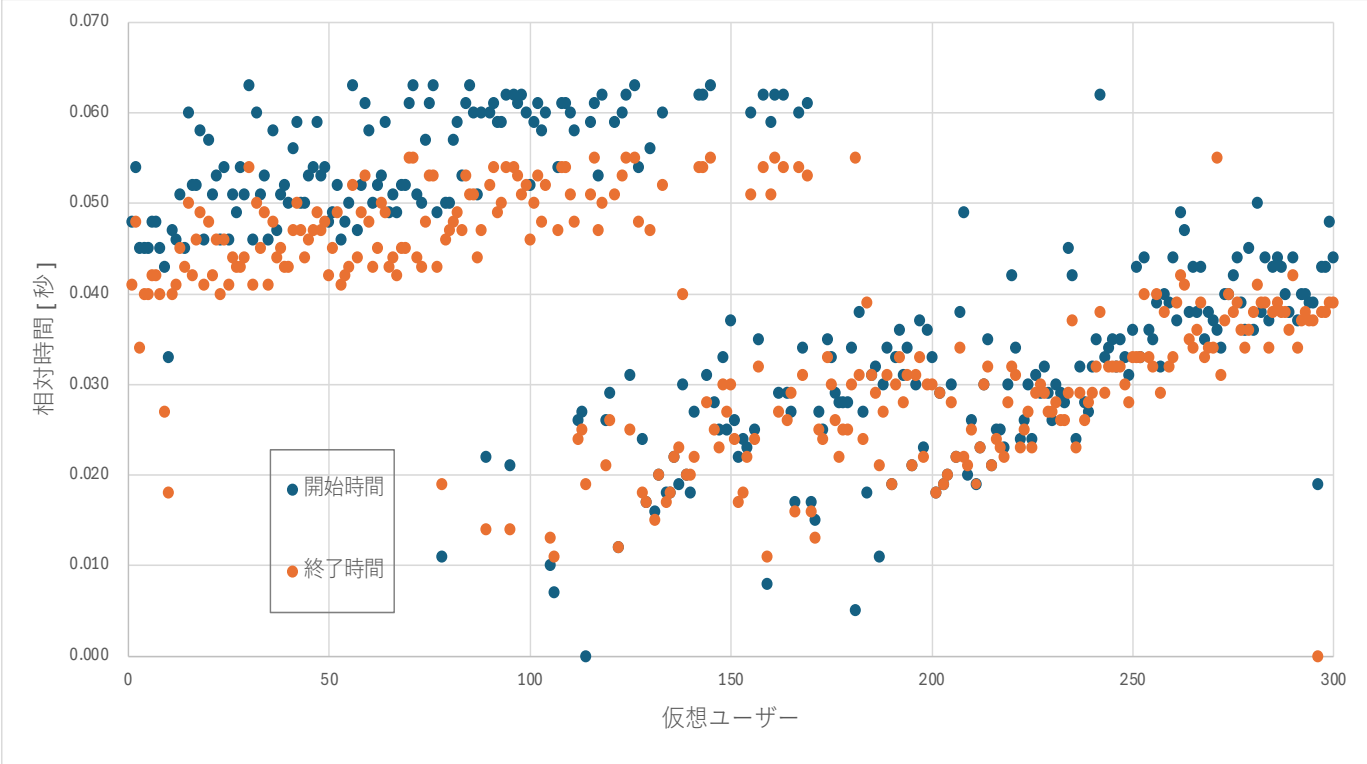


図 19　仮想ユーザー(300 人)のリクエスト開始・終了時間

仮想ユーザー100 人, 300 人のリクエスト時間およびリクエスト開始・終了時間で分かったことを以下に示す.

[リクエスト時間]

100 人：全ての仮想ユーザーが 150.130 秒以内である.

300 人：全ての仮想ユーザーが 150.160 秒以内である.

[リクエスト開始・終了時間]

100 人：仮想ユーザー識別番号が大きくなるほど開始・終了時間が共に遅れており, グラフは右上がりになっている.

300 人：開始・終了時間が共に右上がりになっている区間が 2 箇所ある.

第 6 章　評価

5. 実験のリアルタイム性測定実験と負荷耐性測定実験の結果をもとに RTTD システムの評価を行う.

6.1 リアルタイム性の評価

Jacob Dreyer は LinkedIn 内でリアルタイムシステムにおけるレイテンシの期待値を以下のように解説した.

「The actual latency in a given case depends on the geographical location of producer, server and client and also somewhat on data frequency and volume. Typical values are 20-200ms. *Milliseconds* that is! [13)]」

訳：特定のケースにおける実際のレイテンシは，プロデューサー，サーバー，クライアントの地理的な場所，およびデータの頻度と量によって多少異なる．一般的な値は 20〜200 ミリ秒である．つまり，ミリ秒単位である．

本研究の RTTD システムのレイテンシは 232.52 ミリ秒である．Jacob Dreyer が話す 20〜200 ミリ秒の範囲には収まっていないが，ミリ秒単位という想定には該当する．

続いてジッターを評価するが，その前に図 11 に地上局のデータ到達時間間隔の変動を加えた散布図を図 20 に示す．すなわち，地上局とブラウザのデータ到達時間間隔の変動は近似していることが分かる．よって，ジッターには衛星局の動作ロジックや IM920sL によるデータ損失が大きく影響していることが分かる．

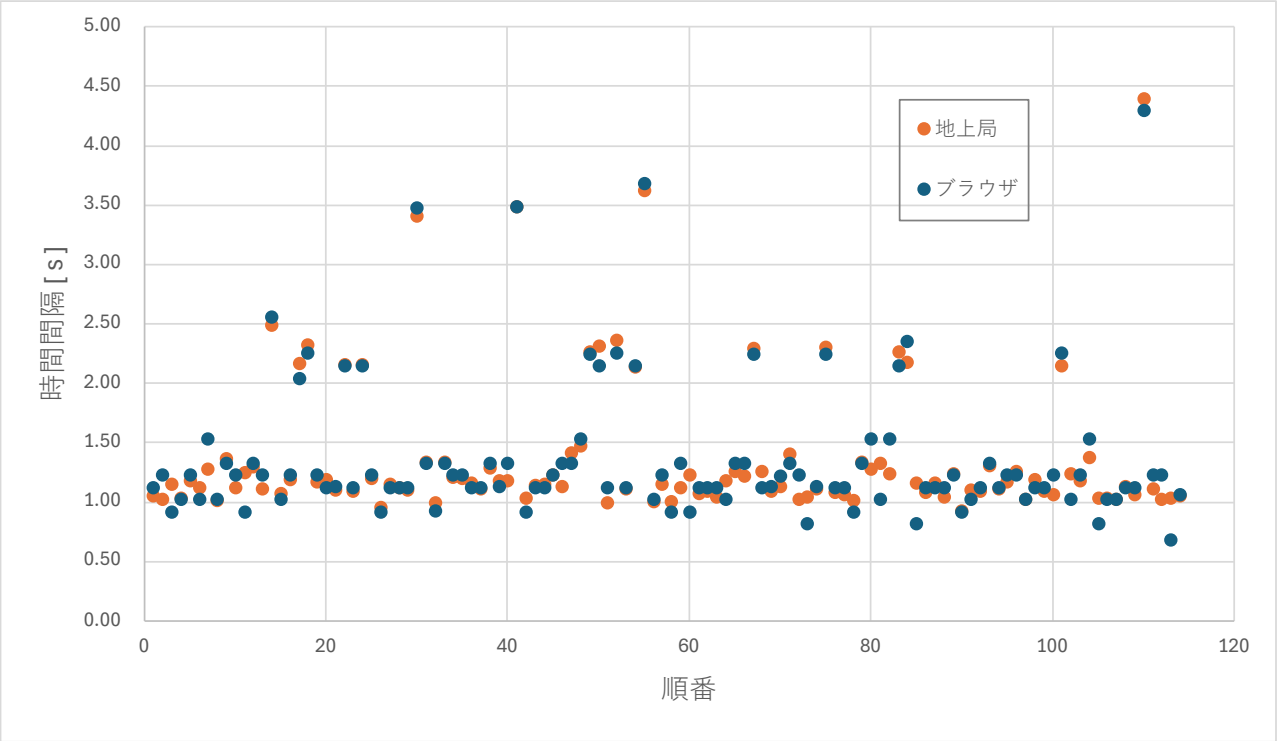


図 20　データ到達時間間隔の変動（試行 1）

6.2 負荷耐性の評価

まず, RTTD システムに接続可能なクライアント数が約 450 人であることを評価する. 打ち上げ時に RTTD システムを使用する可能性がある人数は，打ち上げスタッフや RTTD システム使用を許可された外部の人などを含めて約 100 人未満と想定している. そのため，打ち上げ実験本番では全てのクライアントに対して正常に動作する見込みがある. また，ネットワーク環境を見ると，負荷耐性測定実験ではスループットは 123.1Mbps(Uplink),72.1Mbps(Downlink)である. 鹿児島県の携帯キャリアのスループット平均が 19.54Mbps(Uplink),72.73Mbps(Downlink)[14]であるため，Downlink を見ると負荷耐性測定実験は打ち上げ実験時の各クライアントの想定されるネットワーク環境と一致する（RTTD はバックエンドサーバーへのリクエストが一つに対してレスポンスが長期的かつ多数であるため Uplink のスループットの違いは考慮しない）. よって，負荷耐性測定実験においては本来（打ち上げ実験時）一人で使用する Downlink 帯域を多数で使用していることによってクライアント側のネットワーク内で限界が生じ

て約 450 人という結果になったと考える．そして，鹿児島大学内で eduroam[14]（258.9Mbps(Uplink), 194.4Mbps(Downlink)）を使用して図 14 と同様の条件で負荷耐性測定実験を行うと図 21 の結果になった．仮想ユーザー約 5000 人まで接続可能であった．なお，実験後 EC2 にアクセスしバックエンドサーバーの残りメモリ使用可能量を見るとまだ残っていたため，バックエンドサーバー自体の最大接続クライアント数は 5000 人より更に多いことが考えられる．

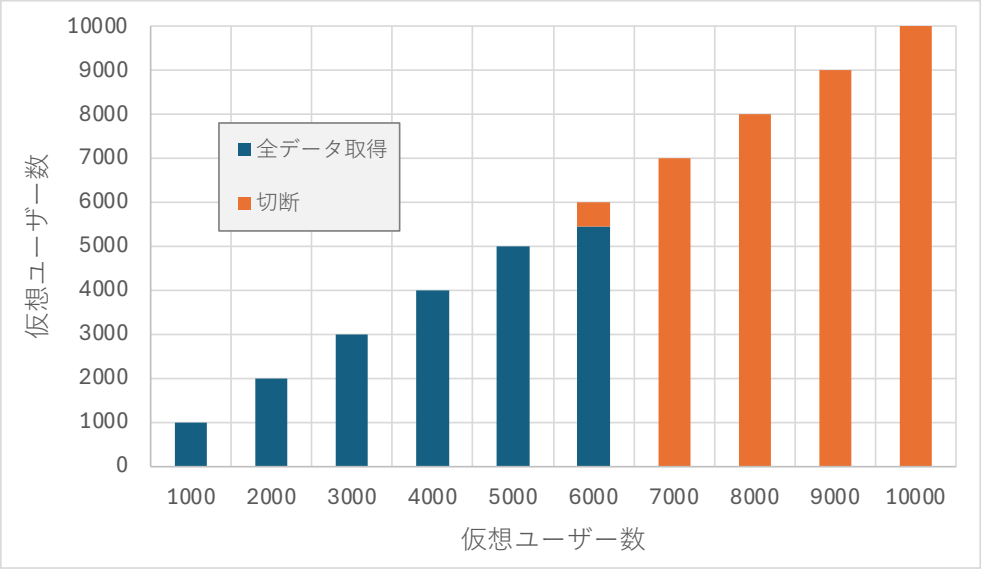


図 21　負荷耐性測定実験結果（eduroam）

次に，リクエスト時間とリクエスト開始・終了時間を評価するために RTTD システム仕様上の事前想定を以下に示す．

図 8 に示したバックエンドのリクエスト処理ロジックを見ると REST リクエストが来た順にリスト B に位置情報送受信用チャネルを入れる．そして gRPC リクエストを受信後，位置情報送受信用チャネルに送信する順番はリスト B 内のチャネル 1 が最初でチャネル N が最後である．そして，負荷耐性測定実験では図 14 に示した通り仮想ユーザーが同時に REST リクエストを行うが，プログラム上では JavaScript の Promise.all(引数：リスト)メソッドを使用しているため完全な並行処理は行うことはできない．よって実際は引数のリスト内の値が順番に実行される．その順番が仮想ユーザーの識別番号でもある．つまり，バックエンドのリクエスト処理ロジックと負荷耐性測定実験で使用したプログラムから，仮想ユーザーの識別番号が小さいほどリクエスト開始・終了時間が共に早く，識別番号が大きいほどリクエスト開始・終了時間が遅くなる

はずである．なお，リクエスト時間はリクエスト開始・終了時間が相対的に同じ時間ずれるだけなのでほとんど変化しないはずである．

上記の事前想定をもとに図 16~19 を見る．リクエスト時間は仮想ユーザーが 100 人と 300 人において共に横ばいのグラフになっているためほぼ変化していない．よって，事前想定通りの結果になっている．リクエスト開始・終了時間は仮想ユーザーが 100 人の場合においては開始・終了時間共に右上がりのグラフになっている．しかし，仮想ユーザーが 300 人の場合は識別番号が小さいほど開始・終了時間が早いという結果にはなっておらず，加えて右上がりの区間が 2 箇所存在している．よって，仮想ユーザーが 100 人の場合は事前想定の結果になったが，300 人の場合は事前想定の結果にならなかった．

第 7 章　結論

本研究では小型ハイブリッドロケット打ち上げのミッションの 1 つとしてリアルタイムロケット軌道表示（RTTD）システムを開発した. RTTD システムの構成は衛星局, 地上局, インフラストラクチャー, バックエンド, フロントエンドであり, 各実装ロジックを示した. 実装した RTTD システムを用いたリアルタイム性測定実験では引用したレイテンシの目標値以下には収まらなかったが, 近い値を残すことはできた. 負荷耐性測定実験では打ち上げ実験時に RTTD システムを使用する可能性がある人数より多くのクライアント数が接続可能であったため, 運用においては十分に要件を満たしていた. また, 各ユーザーのリクエスト時間はほぼ一定で実装通りの結果になっていたが, リクエスト開始・終了時間は想定外のグラフ推移となった部分があるため原因解明を行う必要がある.

謝辞

本研究を進めるにあたり, ご指導していただいた福島誠治教授に感謝を申し上げます. そして, 忙しい中, 実験の手伝いをして頂いた福島研究室の学生の皆様にも感謝を申し上げます. また, 様々な制限がある中での打ち上げ実験のスケジュール調整を先導して頂いた片野田洋教授および片野田研究室の方々, 人工衛星筐体の作製に取り組んで頂いた藤田ワークス様にも感謝を申し上げます.

参考文献

1) 片野田 洋, 永田 晴紀, “ハイブリッドロケットの c * 効率について”, 鹿児島大学工学部研究報告 第 58 号（2016）, p.1, https://www.eng.kagoshima-u.ac.jp/wp-content/uploads/2018/03/H28_report_01.pdf
2) GRPC Core 45.0.0, “g_stands_for”, https://grpc.github.io/grpc/core/md_doc_g_stands_for.html
3) GORM, “データベースに接続する”, https://gorm.io/ja_JP/docs/connecting_to_the_database.html
4) Amazon AWS Documentation, “Amazon VPC とは？”, https://docs.aws.amazon.com/ja_jp/vpc/latest/userguide/what-is-amazon-vpc.html
5) Amazon AWS Documentation, “リージョンとゾーン”, https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/using-regions-availability-zones.html
6) Amazon AWS Documentation, “アマゾン EC2 とは？”, https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/concepts.html
7) Amazon AWS Documentation, “Amazon Relational Database Service (Amazon RDS）とは”, https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/UserGuide/Welcome.html
8) Amazon Web Service, “Amazon Route 53”, https://aws.amazon.com/jp/route53/
9) お名前.com, https://www.onamae.com/
10) Amazon AWS Documentation, “AWS Certificate Manager とは？”, https://docs.aws.amazon.com/ja_jp/acm/latest/userguide/acm-overview.html
11) Amazon AWS Documentation, “Application Load Balancer とは？”, https://docs.aws.amazon.com/ja_jp/elasticloadbalancing/latest/application/introduction.html
12) Amazon AWS Documentation, “EC2 Instance Connect エンドポイントを使用したインスタンスへの接続”, https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/connect-with-ec2-instance-connect-endpoint.html
13) LinkedIn, “Latency in real-time systems”, https://www.linkedin.com/pulse/latency-real-time-systems-jacob-dreyer/
14) eduroam, “eduroam JP の概要”, https://www.eduroam.jp/about