令和２年度　特別研究報告書

アドホックネットワークを用いた渋滞情報の伝達

龍谷大学　理工学部　情報メディア学科

T170469　川本　涼太

指導教員　三好　力　教授

内容梗概

渋滞や交通規制などの道路交通情報をドライバーに知らせる方法の 1 つとして、渋滞情報取得システム VICS が利用されている。道路交通情報はカーナビゲーションシステムや FM 多重放送などを介して知らせることができ、ナビゲーションの渋滞回避や到着時間の予想などに役立てられている。しかし、受信のタイミングや状況によっては 5 分から 10 分のタイムラグが生まれ、実際の渋滞との差が生まれてしまう。

本研究では、MANET を用いてタイムラグの問題を解決する方法を提案する。実験では、本研究で提案する情報の伝達において、先行車を判定するアルゴリズムが有効であるか確認した。実験の結果、先行車の判定を行うことが出来た。

目次

第1章 はじめに

1.1 研究目的

渋滞や交通規制などの道路交通情報をドライバーに知らせる方法の 1 つとして、渋滞情報取得システム VICS が利用されている。VICS を利用できるカーナビゲーションシステムは増えており、2019 年ではカーナビゲーションシステムの出荷台数のうち VICS 対応の車載器は約 71%になる。カーナビゲーション及び VICS 車載器年別出荷台数を図 1.1 に示す。グラフは 2009 年 1 月から 2020 年 4 月までのデータである。

道路交通情報はカーナビゲーションシステムや FM 多重放送などを介して知らせることができ、ナビゲーションの渋滞回避や到着時間の予想などに役立てられている。近年ではスマートフォンが普及し、Google マップのナビゲーションモードなどのナビゲーションアプリを利用して目的地まで移動する人が増加している。自動車メーカーではディスプレイオーディオを設置するだけでナビ機能は搭載せず、「Android Auto」や「Apple CarPlay」にナビ機能を担わせることで車両価格を抑えている傾向がある。

Google マップでは、匿名で収集したユーザーデータ、交通センサーデータ、衛星データに基づいて、絶えず情報を更新することにより交通状況をドライバーに伝えている。VICS では、都道府県警察や道路管理者からの情報や交通センサーデータを収集して処理・編集を行いナビゲーションへの反映や FM 多重放送を通じて交通状況をドライバーに伝えている。しかし、受信のタイミングや状況によっては 5 分から 10 分のタイムラグが生まれる。実際の交通状況では交通量の変化が激しい場所では渋滞の変化も激しくなるため、タイムラグがあると実際の渋滞との差が生まれてしまう。

本研究では、MANET を用いてタイムラグの問題を解決する方法を提案する。MANET は端末同士でのネットワークを構築することができるため、渋滞情報を VICS よりも早く反映することが期待できる。このシステムを VICS の+αとして用いることで、目的地までのルート選択の判断材料を増やすことが出来る。また、VICS が設置されていない道にも対応できる。

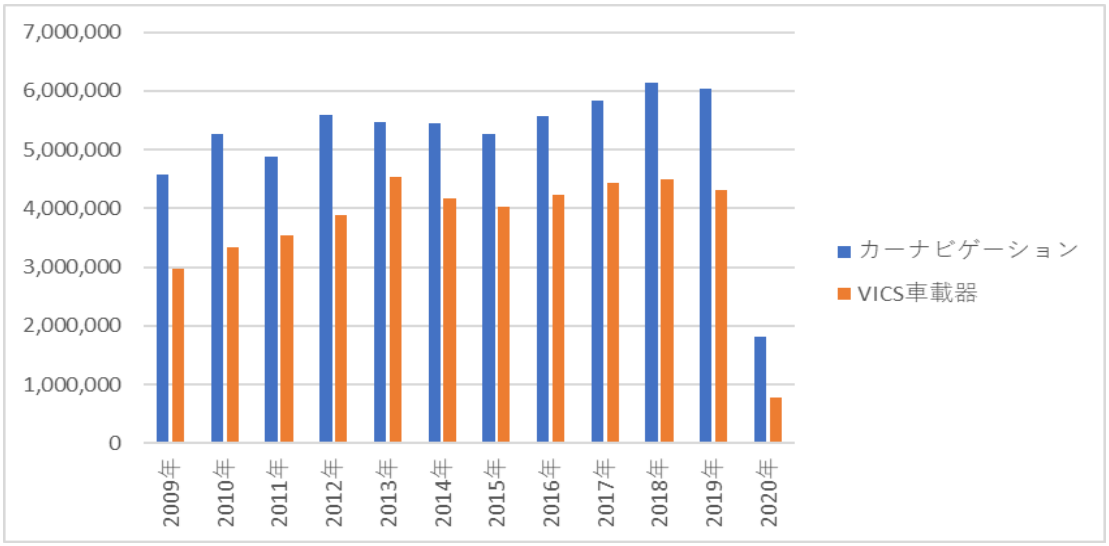

図 1.1　カーナビゲーション及び VICS 車載器年別出荷台数

第2章 既存技術

2.1 MANET

モバイルアドホックネットワーク（MANET:Mobile Ad Hoc Network,以下 MANET）は、携帯端末などの通信機器（ノード）の集まりであり、ノード同士で一時的な通信ネットワークを構築することにより相互通信が可能になる。ノード同士でのネットワークのため、通信基地局、無線 LAN のアクセスポイント（AP）、基地局-AP 間のバックボーンネットワークを必要とせず、場所を選ばずに即座にネットワークが構築できる。

ノードには送信元と宛先がある。この 2 つのノードが近在し、互いに通信範囲内である場合にはノード間で直接通信が可能となる。このような通信形態を 1 ホップ通信と呼ぶ。また、1 ホップ通信で通信可能なノードを隣接ノードと呼ぶ。送信元と宛先間の距離が長い場合や通信範囲外である場合は、間に他のノードを経由して通信を行う。このような通信方式をマルチホップ通信と呼ぶ。経由されるノードは中継ノードと呼び、ルーターの役割を果たしている。1 ホップ通信とマルチホップ通信を図 2.1 に示す。

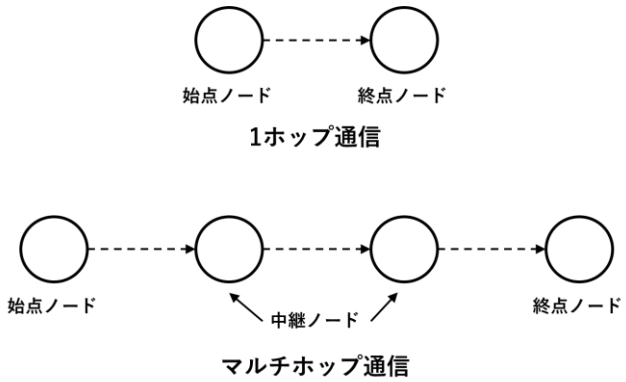

図 2.1 1 ホップ通信とマルチホップ通信

ノードは移動する可能性があり、ノード相互間のリンクが不確実であるため、通信経路の決定、無線周波数の帯域や出力などの技術的課題がある。しかし、災害時において迅速なネットワークの構築ができるため、被災地における状況把握、情報提供、連絡手段の確保が期待できる。また、ITS（高度道路交通システム）や自動運転などに関わる分野でも期待されている。[1]

2.1.1 VANET

車々間通信に適用されるアドホックネットワークを VANET（Vehicular Ad hoc NETwork,以下 VANET）と呼ぶ。近年自動運転への技術が進歩し、自動車にはセンサーやレーダー、3D マップなどを用いたナビゲーションシステムが搭載されている。これらを利用することで車々間通信では,自動車同士の交通事故防止や周辺の交通

状況把握のため有望な技術とであると考えられる。

車には多くのセンサーが搭載されており、それらが車々間通信でやり取りされる情報の発信源になる。車々間通信は車同士の距離が近く、匿名の車の間で行われることが多い。VANET はこのような条件やシナリオを想定しており、次のような利点がある。[2]

- 安全に関する情報の許容時間内の低遅延転送
- 低通信コスト
- 位置情報に基づく近隣車への通信

道路上のセンサーや信号機などを固定ノードとすることにより、インターネットなどの固定網のゲートウェイとして VANET に参加させることが可能である。また、車々間通信、路車間通信における情報の一時的な保管場所として利用も可能である。

2.2 VICS

VICS(ビックス)とは、事故や工事による渋滞や交通規制などの道路交通情報を、FM 多重放送やビーコンを用いてカーナビゲーションシステムに反映させることでドライバーに知らせるシステムである。VICS 情報は 24 時間 365 日提供され、カーナビゲーションシステムによるルート検索や渋滞回避に活用されている。[3]

2.2.1 VICS の情報収集

都道府県警察や道路管理者が交通センサーにより収集した渋滞情報、旅行時間情報や管理者等が入力した交通規制の情報を日本道路交通情報センターが道路交通情報システムにオンラインにより収集する。(図 2.2) その情報を基に VICS がデータの処理、編集を行っている。

図 2.2　情報収集の流れ

2.2.2 VICS の情報提供

VICS の情報は FM 多重放送や高速道路では電波ビーコン、主要な一般道では光ビーコンを用いてナビなどの車載器に向け情報提供されている。渋滞情報や事故、故障、工事、災害、気象条件等による規制情報をはじめ、電波ビーコンはインターチェンジ間の所要時間など、光ビーコンは駐車場の満車・空車の情報などを提供している。

2.2.3 VICS WIDE

VICS WIDE では VICS の機能に加え、タクシーなどのプローブカーから位置や速

度、時刻などのデータ（走行軌跡）であるプローブ情報を受け取り、交通センサーがない区間の交通情報も得ることが出来る。プローブカーはタクシーなど公共性の高い乗り物が用いられるが、トヨタ、ホンダ、日産などの車に搭載されている車載器からのプローブ情報を利用して、より多くの情報を収集する実証実験が 2020 年 4 月より行われている。（図 2.3）

ルート案内ではリンク旅行時間を考慮することで複数のルートごとの所要時間が分かるようになり、より精度の高い渋滞回避が可能である。リンク旅行時間とは道路の交差点間などの一定区間の通過に要する時間である。距離・走りやすさに加えて 5 分ごとに更新されるリンク旅行時間を考慮してルート検索を行うことで、VICS WIDE は従来の VICS よりも到着予想時刻のズレ改善やルートの最適化ができる。[3]

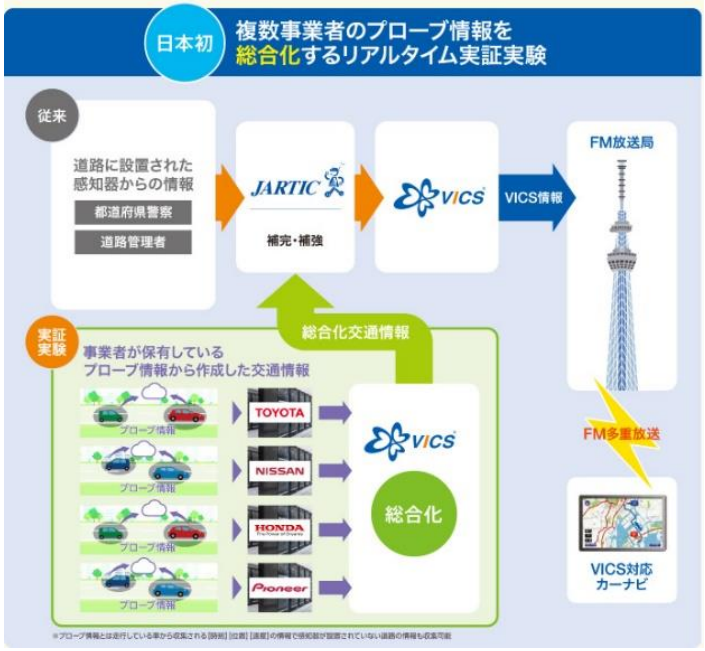

図 2.3　プローブ情報を総合化する実証実験

2.2.4　VICS の問題点

VICS の問題点として 2 点挙げられる。1 点目は、情報収集、通信、処理、編集の関係で、5 分から 10 分程度のタイムラグが生まれる点である。渋滞状況は刻々と変化するため、情報での渋滞の最後尾とリアルタイムでの渋滞の最後尾の位置がタイムラグによりズレが生じてしまう。すると、想定よりも渋滞が長くなるという事態がおこり、渋滞に巻き込まれる可能性がある。2 点目は情報を得ることが出来ない区間がある点である。VICS では都道府県警察や道路管理者の交通センサーが設置されていない区間では渋滞状態を監視し続けることは出来ない。VICS WIDE では交通センサーがない区間でもプローブカーにより情報が得ることができるが、プローブカーが通過しない区間の情報は得られない。また、車載器からデータを送るために携帯回線を利用しているため利用料金がかかることや、災害時に通信基地局が

停止すると回線が途切れ、利用できなくなる可能性があるという問題点がある。

本提案手法では、MANET により情報を提供するため、リアルタイム性が高く渋滞をより正確に検知することが期待できる。また、ノード間の通信手段には携帯回線を使用しないため、利用料金が掛からない。そして、利用する設備や機材が少ないため、整備費用などのコスト削減が期待できる。

2.3 Google マップ

Google マップは Google が提供する地図アプリである。Google マップでは、店や建物の情報、車や徒歩のナビゲーション、電車の乗り換え案内などを行うサービスを提供しており、旅行や仕事などで土地勘が無い場所を訪れる際などに利用されている。

2.3.1 Google マップのナビゲーションについて

Google マップには、車のルート案内を行うサービスが提供されている。ナビゲーションを開始すると、ルート案内に加えて渋滞情報や交通規制情報、到着予想時刻などの情報が表示され、車に搭載されるナビゲーションシステムとほぼ同様の機能が備わっている。これらの機能は無料で利用でき、地図更新にも料金がかからないため、車にカーナビゲーションシステムを設置せずアプリを利用することで代用する人が増えてきている。

Google マップは現在地を特定するために次のような方法を用いている。スマートフォンなどの端末に搭載されている GPS を使用して推定誤差 20m 以内で現在地を特定する方法。近くの店などの Wi-Fi ネットワークを使用して現在地を特定する方法。携帯ネットワークに接続している場合、電話通信基地局から現在地を最大誤差数千m以内で現在地を特定する方法。[4]

渋滞を検知する方法として、Google マップのナビゲーションはアプリを利用して運転しているユーザーの位置や車速などのデータを集める。利用者の複数台が低速走行を行っている区間は渋滞区間としてマップ上に道が赤く表示され、他のドライバーに渋滞を知らせている。再び通常速度で走行する車両が複数台現れると、渋滞は解消されたと判断し、表示を取り消す。また、過去の交通データパターンと現在の交通データで機械学習を行い、双方のデータに基づいた近い未来の渋滞を予測している。

その他の交通情報を取得するために、地方自治体から提供されるデータやユーザーから提供されるデータを利用している。ユーザーから提供されるデータは、ドライバーが運転中に得た道路情報を「レポートの追加」機能（図 2.4）で Google に報告することができる。この機能は、道路や車線の閉鎖、道路工事、事故車両、落下物などをユーザーがいち早くナビゲーションに反映することができる。そのため、土砂崩れ、吹雪、その他の自然災害などによる道路状況の予期せぬ変化の把握にも

役立てられる。

図 2.4 「レポート追加」機能

2.3.2 Google マップの問題点

Google マップの問題点として、ユーザーがデータを取得、提供する際にデータ通信量が発生することである。あらかじめルート案内を Wi-Fi 環境下などでダウンロードしておき、ルート案内時はオフラインで利用することでデータ通信を行わない方法もある。しかし、本研究では渋滞状況の変化をリアルタイムに伝えることを目指しているため、Google マップで常に最新の交通情報を得るためにはルート案内時にデータ通信を行う必要があり、屋外での利用には携帯電話会社との契約が必要になる。

他にも、最短距離でルート検索をしているため、ルート案内の際に狭い道や住宅街などの入り組んだ道を選択することがあることや、建物の裏側に案内されることがあるなどの問題点が挙げられる。

本提案手法では、MANET を用いてノード同士のネットワークを構成することにより、データ通信を行わずに最新の渋滞情報を取得できる。そのため、携帯電話会社と契約する必要が無く、利用料金が発生しないため、導入し易くなると考える。

2.4 Android 端末の GPS の誤差について

先行研究では Android 端末を用いて GPS によって位置情報と速度情報を取得し

た時の誤差について検証する実験を行っている。[5]

実験結果として位置情報は、端末が動いていない状況での測定は位置情報にバラつきが少ない。しかし、端末が移動している状況で測定を行うと、位置情報のバラつきがある。この原因として、移動速度を一定にすることの難しさや Android 端末の処理速度が移動速度に追いついていないなどの理由が挙げられている。そのため、実際の移動距離と位置情報から算出した処理では最大 7m の誤差があったとしている。

速度情報の取得については、実際の速度と位置情報から算出した速度には大きな誤差はないとしている。

この先行研究より、本研究ではノードが移動している可能性は十分に考えられるため、GPS により取得した位置情報には誤差が生まれると考える。

第3章 提案手法

従来の VICS では、情報の反映に 5 分から 10 分のタイムラグが生じる。また一般道路においては、主要な道路のみにしか対応していない。そこで本研究では、一般道路において MANET を使用し、スマートフォンを用いて渋滞情報をリアルタイムに取得し後続車に情報を伝達するアプリを制作する。このアプリを用いて渋滞検知を行い、MANET を用いてリアルタイムでドライバーに知らせることを目的とする。MANET を用いることで携帯回線を用いずに周囲の端末と通信を行い、データ通信による携帯利用料金をかけないようにする。しかし、位置情報を取得する際にスマートフォンの GPS 機能だけを用いると得た情報に誤差が出てしまう。本研究では、位置情報を絶対的な位置と捉えるのではなく、相対的な位置情報と考えて扱うこととする。相対位置情報では 2 つのノードの位置関係を距離や方角によって知ることができ、端末の GPS 情報に含まれる絶対的な誤差が類似した原因による場合、ノード間の距離が近いほどノード間の相対的な誤差は小さくなると考えた。端末が位置情報を取得するとき、通信する衛星の数や通信環境によって誤差が生まれる。ノード同士の距離が近い場合は周辺の通信環境が似ているため、誤差の出方も似たものになる。そのため、ノード同士の位置関係や進行方向を把握する上では絶対位置情報の精度を求めなくても可能であると考える。

将来的に車載器を用いてアドホック通信を行えるようになると、より精度の高い位置情報や車両情報が得ることが出来るため、精度の高い渋滞情報の伝達が期待できる。

3.1 提案手法のアルゴリズム

日本道路交通情報センターでは渋滞・混雑状態を次のように定義している。高速道路では、時速 40 ㎞以下の状態または、停止・発進を繰り返す状態が 1㎞ かつ 15 分以上継続した状態を渋滞とする。一般道路では、時速 10㎞ 以下の状態を渋滞とし、時速 10㎞ から時速 20㎞ は混雑状態とする。

走行中の車両が渋滞または混雑の定義に当てはまる状態のとき、そのノードは周囲に状態の情報と位置情報、走行履歴を基にした進行方向の情報を送信する。情報を受け取ったノードは、進行方向が一致しているか、情報の送信元が走行軌跡上にいないかを基に先行車両であるか判断し、先行車である場合は情報をナビゲーションに反映させる。後ろを走行する車両に情報を伝えるため、中継ノードの役割を与えることにより、距離が離れたノードにも情報を伝達できるようにする。そのため、後ろを走る車両は先の交通状況を確認することができる。

渋滞・混雑を検知、情報を中継、情報を受信、先行車を判定の 4 つのアルゴリズムを図 3.1.1 から図 3.1.4 に示す。

3.1.1 渋滞・混雑の検知

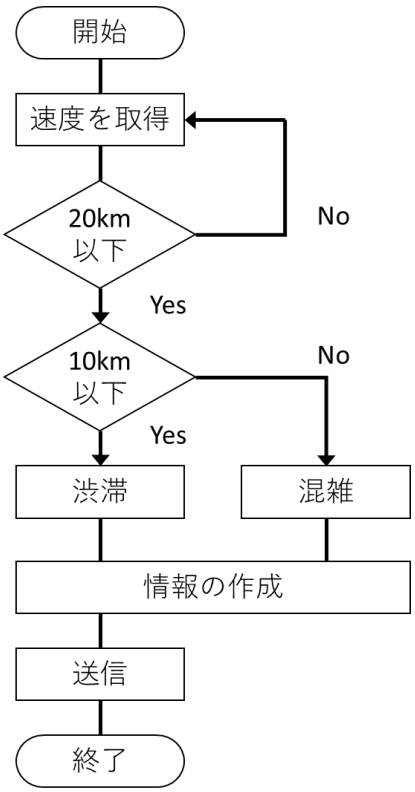

図 3.1.1 渋滞・混雑の検知

車両は常に自車の速度を計測しており、ノードの状態が定義に当てはまっている間は常に情報を送信し続ける。状態は 2 種類あり、時速 10km 以下の場合は「渋滞」、時速 10km から 20km の場合は「混雑」としている。当てはまらなければ何も行わないことで、ネットワーク内のデータ通信量を削減し、混雑を回避する。

3.1.2 情報の中継

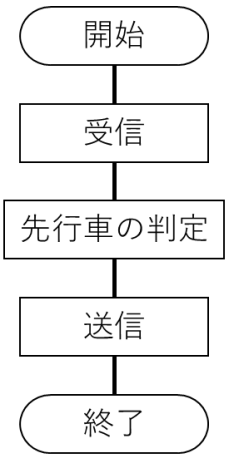

図 3.1.2 情報の中継

先行車の情報を後続車へ伝えるため情報の中継を行う。いつ情報が送られてくるのか分からないため、ノード間で相互通信が可能である状態のときは常に受信できる体制をとる。受信した時、先行車であるかの確認を行う。先行車の判定については 3.1.4 に記載している。先行車である場合はバケツリレーの要領で周囲に情報を送信する。先行車でない場合は、送信元の進行方向が異なる場合か後続車である場合のため何も行わず中継しない。

3.1.3 ナビゲーションへの反映

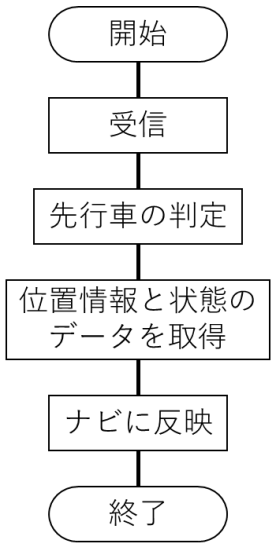

図 3.1.3 ナビゲーションへの反映

受信した情報は、ナビゲーションに情報を反映させることによりドライバーに通知する。情報を受信した場合、先行車であるか確認を行う。先行車である場合は状態、位置の情報をナビゲーションに表示し、渋滞・混雑の範囲を通知する。

このアルゴリズムを実行している時、後方車両にも伝える必要があるため 3.1.2 情報の中継のアルゴリズムと平行して実行しなければならない。

3.1.4　先行車の判定

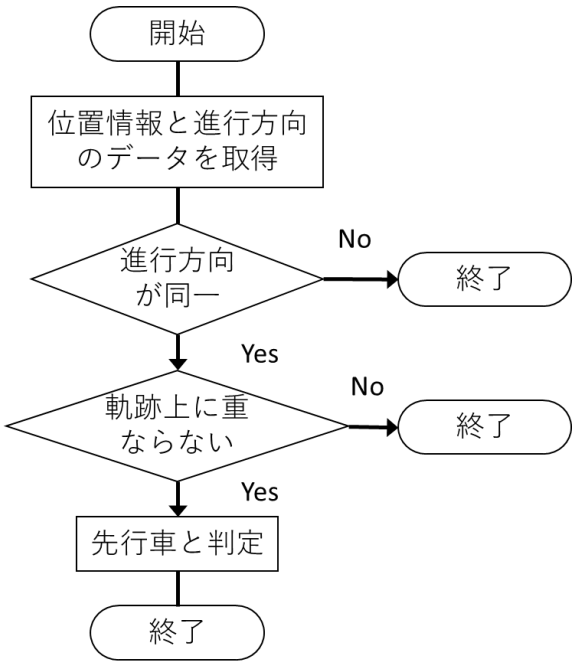

図 3.1.4 先行車の判定

走行中に受信する情報は必ずしも先行車両の情報とは限らず、反対車線を走る車両や交差点などで別方向に向かう車両、後方の車両といった可能性がある。アドホック通信には指向性が無いため、情報を送信または受信する方向を限定することが不可能である。そのため、情報を受け取った時に先行車からの情報であるかを精査することが必要である。

まず、受信した情報から位置情報と進行方向のデータを取り出す。その情報と自車の進行方向と一致しているか確認、自車の走行軌跡と重なっていないかを確認する。もし、位置情報が自車の軌跡上にある場合は後方に走る車両からの情報であるため、先行車でないと判断する。

第4章 実験

4.1 実験概要

受け取った情報が先行車の情報であるかの判断を行うシステムでシミュレーションを行う。作成したアプリに疑似情報を入力し、あらかじめ決められた情報と照らし合わせることで、送信元との位置関係と進行方向から先行車を判定し、その情報を中継するべきかどうかを判断する。

この検証により、本研究で提案する情報の伝達において、先行車を判定するアルゴリズムが有効であるか確認する事を目的とする。

4.2 アプリケーション

今回作成したアプリは自車の進行方向や位置などの疑似の条件を設定しておき、その条件と入力された条件を比較することで先行車の判定を行い、先行車からの情報であると判定した場合はデータを送信するためのアルゴリズムに移行する。

アプリ起動時の画面を図 4.2 に示す。Latitude、Longitude の欄には送信元の位置情報を疑似座標の緯度経度で表し入力する。北緯を Latitude、東経を Longitude で表す。疑似座標の詳細については図 4.3 シミュレーション環境にて説明している。

Direction の欄には、送信元の進行方向を入力する。北向きを「N」、南向きを「S」と表す。本来 GPS を用いて進行方向を計算する場合、移動する前と後の座標の差から求めるが、今回は疑似座標の差から求めたと仮定して仮の進行方向を与えている。

State の欄には、送信するデータを格納する。本研究では渋滞情報として仮に「Traffic jam」を入力する。

今回作成したアプリは自車の位置情報と Direction の情報、送信車両の位置情報と Direction の情報、この 4 つの情報で比較を行い先行車の判定を行う。入力された送信元の位置情報と Direction の情報と自車の位置情報と Direction の情報と合う場合、表示画面には位置情報、Direction、State の入力情報がそのまま表示される。State の欄に情報が表示された時、アプリは情報を中継すると判断し、中継するアルゴリズムに移行することを示す。入力された位置情報もしくは Direction の条件が自車の条件と異なる場合、受け取った情報は先行車の情報では無いと判断するため State は表示しない。つまり後続車に伝達しないことになる。このように先行車の判定を行い受け取った情報を周囲の車両に配信するかを判断する。

このアプリは文字列で表した擬似的な進行方向と位置情報を文字判別により判断している。進行方向は、あらかじめ決めておいたアプリ内部にある情報と送信元の情報を照らし合わせ判断している。位置情報を擬似的に文字で表すことが難しいため、あらかじめ疑似座標を用いて各車両の位置情報を決めておき、送信元の位置

情報と比較できるようにしている。そのため今回の実験では、周辺車両から情報を受け取ったと仮定してアプリを使用する人が代わりに送信元車両の位置情報を疑似座標で表し入力する。送信元の車両が自車の前方あるいは後方にいる想定の情報であるという条件の下、入力された条件と内部の条件を照らし合わせ判断する。

図 4.2　アプリ起動時

4.3 シミュレーション環境

シミュレーションは片側 1 車線の一般道路で情報の中継を行う場面を想定して行う。車両は A から E の 5 台があり、それぞれの車両には渋滞情報を発信するための端末が備わっている。車両の位置関係と進行方向を表したシミュレーション環境のイメージ図を図 4.3 に示す。シミュレーション環境には 3×3 の疑似座標があり、それぞれの車両に位置情報[北緯, 東経]が与えられている。枠で囲まれている車両 C を自車とし、位置情報を[35.1, 135.0]、進行方向を「N」と設定する。自車の同一車線前方には車両 B が走行しており、後方には車両 D が走行している。反対車線前方には車両 A が走行しており、反対車線後方には車両 E が走行している。車両 A, B, D, E は渋滞を検知し、周辺車両に向け渋滞情報をブロードキャストにより配信を行っている。送信元の車両は自車の前方または後方にいることを想定している。

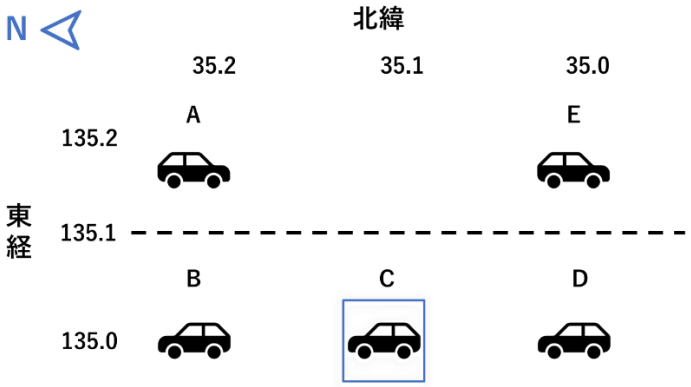

図 4.3　シミュレーション環境のイメージ

4.4　実験内容

今回の実験では車両 C が情報を受信したとき、先行車である車両 B の情報であるかを判断する先行車の判定のアルゴリズムの検証を行う。正確に判定できているか確認するため、他に 3 パターンの受信条件を用意し、合計 4 パターンで比較実験を行う。受信条件のパターンを表 1 に示す。パターン 1 は車両 B から受信するときの条件、パターン 2 は車両 A から受信するときの条件、パターン 3 は車両 D から受信するときの条件、パターン 4 は車両 E から受信するときの条件である。

表 1　受信条件のパターン

	[北緯, 東経]	進行方向	状態
パターン 1	[35.2, 135.0]	N	Traffic jam
パターン 2	[35.2, 135.2]	S	Traffic jam
パターン 3	[35.0, 135.0]	N	Traffic jam
パターン 4	[35.0, 135.2]	S	Traffic jam

4.5　実験環境

実験環境を以下に示す。

実験端末：　Sony Xperia Z4 SOV31
Android バージョン：　7.0
開発ツール：　Android Studio 4.0
使用言語：　Java, Xml

4.6　実験結果

実験結果を図 4.6.1 から図 4.6.4 に示す。実験結果は複数回行った実験のうち代表的なものを示している。図 4.6.1 はパターン 1 の車両 B から受信した場合、図 4.6.2 はパターン 2 の車両 A から受信した場合、図 4.6.3 はパターン 3 の車両 D から受信した場合、図 4.6.4 はパターン 4 の車両 E から受信した場合、それぞれのシミュレーションを行った画面を示している。

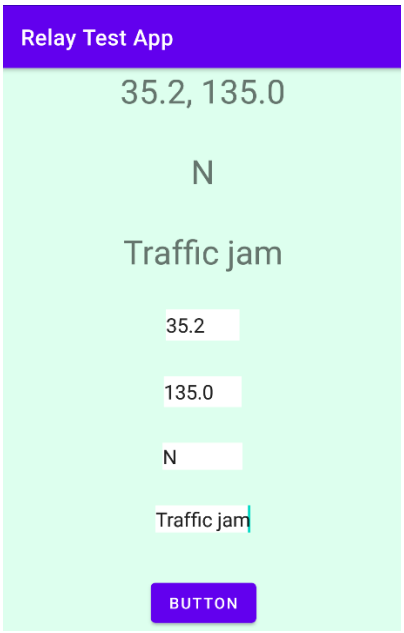

図 4.6.1 パターン 1 の実験結果

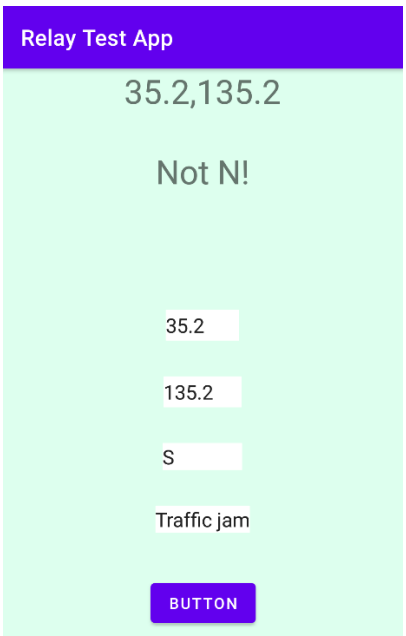

図 4.6.2 パターン 2 の実験結果

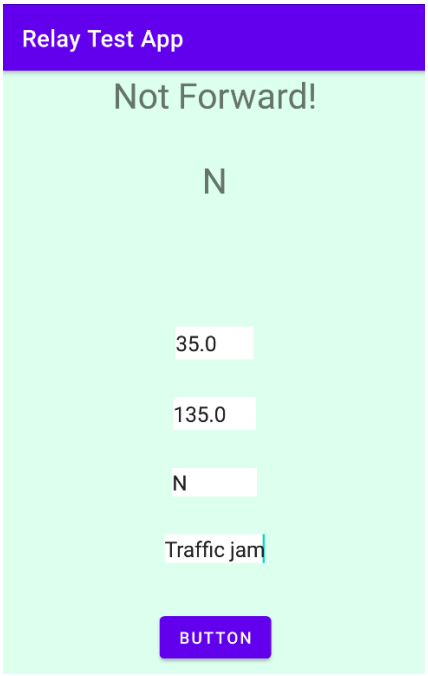

図 4.6.3 パターン 3 の実験結果

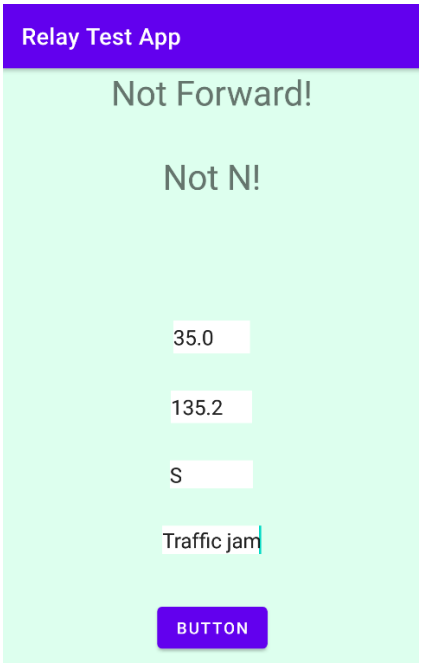

図 4.6.4 パターン 4 の実験結果

4.7 考察

実験結果の図 4.6.1 より、先行する車両 B から受信情報を判断し、渋滞情報を表示することが出来た。図 4.6.2 から図 4.6.4 については後続車や対向車線を走る車両からの情報を判断し、渋滞情報の表示を行わなかった。この実験結果を受け、本研究で提案している先行車を判定するアルゴリズムは先行車からの情報を判断し、

渋滞情報を取り出すことが可能であると確認できた。

本研究では作成していないが、このアルゴリズムにGPSを用いた位置情報の取得と自車の走行軌跡を記憶しておく機能、BluetoothやWi-Fi Directを用いた周辺車両と情報を伝達する機能を組み合わせることにより、情報を受け取り自車の情報と照らし合わせ、伝達するまでの一連の流れが可能になる。

本研究を進める中で出てきた問題点として、中継の終わる基準を決めていない点がある。本研究で提案しているアルゴリズムでは、中継範囲内に通信端末を備えた車両がいる限り渋滞情報が中継され続けるため、中継範囲の制限や時間制限などで渋滞情報を中継しない条件を付け加える必要がある。

第5章 おわりに

本研究では、VICS の情報処理・伝達に起こるタイムラグの問題を MANET により渋滞情報を伝達することで解消する手法を提案した。実験では、先行車の判定を行うアルゴリズムのシミュレーションを行い、先行車の判定を行った後、送信元の渋滞情報を表示するという結果を得ることができた。

今回は、擬似情報を用いた先行車の判定を行い、シミュレーションを行った。しかし、擬似的な情報を用いているため、実際の位置情報のデータを基にして判断しておらず、複数台の端末を車両に乗せ、走行しながら通信するといった実験を行うことができなかった。今後の研究では、複数台の端末を車両に乗せて実際の利用環境と同条件で実験を行うために、渋滞・混雑の検知や情報の中継のアルゴリズムの検証を行う必要がある。渋滞・混雑の検知を行うアルゴリズムでは、スマートフォンで GPS を利用した速度取得の精度の検証、情報を中継するアルゴリズムでは、車々間通信に適した通信方式の検証などが必要である。

今回はスマートフォンなどの端末にアプリをインストールすることで多くの人が利用しやすくしているが、このシステムを車載化することにより、自動車のセンサーと連携を行うことができより正確なデータやその他のデータも取得できるようになることを期待する。

謝辞

本研究を進めるにあたり、多くの助言、指導をしてくださった三好力教授に心から御礼申し上げます。また、研究に関する知識や示唆を頂いた三好研究室の皆様に厚く御礼申し上げます。

参考文献

[1]　電子情報通信学会知識ベース　知識の森　2 章 アドホックネットワーク - 電子情報通信学会知識ベース （2010 年 6 月 10 日）
http://www.ieice-hbkb.org/files/04/04gun_05hen_02.pdf
[2]　無線アドホックネットワーク技術論文特集　車々間通信とアドホックネットワーク　間瀬 憲一
https://search.ieice.org/bin/pdf_link.php?category=B&lang=J&year=2006&fname=j89-b_6_824&abst=
[3]　VICS　一般財団法人 道路交通情報通信システムセンター
https://www.vics.or.jp/
[4]　Google ヘルプ　マップヘルプ　現在地を正確に検出、表示する
https://support.google.com/maps/answer/2839911?co=GENIE.Platform%3DAndroid&hl=ja
[5]　平成 30 年度　特別研究報告書　龍谷大学理工学部情報メディア学科 T150523 西向一哉
[6]　一般社団法人電子情報技術産業協会　民生用電子機器国内出荷統計
https://www.jeita.or.jp/japanese/stat/shipment/
[7]　[Android] スマホに Edit Text を使って文字を入力する
https://akira-watson.com/android/edittext.html

付録

<Main Activity.java>

```
package com.example.releytest1app;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private EditText latitudeNum;
    private EditText longitudeNum;
    private EditText directionText;
    private EditText stateText;

    private TextView directionView;
    private TextView positionView;
    private TextView stateView;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        latitudeNum =
findViewById(R.id.latitude_Num);
        longitudeNum =
findViewById(R.id.longitude_Num);
        directionText =
findViewById(R.id.direction_text);
        stateText = findViewById(R.id.state_text);

        directionView =
findViewById(R.id.direction_text_view);
        positionView =
findViewById(R.id.position_text_view);
        stateView =
findViewById(R.id.state_text_view);

        Button button = findViewById(R.id.button);

        button.setOnClickListener( v-> {
            // エディットテキストのテキストを取得
            String d_text =
directionText.getText().toString();
            String s_text =
stateText.getText().toString();
            // 取得した内容を文字列として認識
            String strlat =
latitudeNum.getText().toString();
            String strlong =
longitudeNum.getText().toString();
            // 入力されたのは数字なので，数字になおす
            double get_lat =
Double.parseDouble(strlat);
            double get_long =
Double.parseDouble(strlong);

            judge(d_text, get_lat, get_long, s_text);
        });
    }

    /**
     *先行車を判断するメソッド
     *  先行車でないと判断した場合、state を返さない
     * @param d_text Direction 進行方向  この端末は北
向き" n "に設定
     * @param get_lat latitude  緯度  入力された緯度
     * @param get_long longitude 経度入力された経度
     * @param s_text state  状態  後続車に伝える内容
```

```
     */
    private void judge(String d_text, double get_lat,
double get_long, String s_text){
        // 自車両のデフォルト値の設定　進行方向 N　座
標(35.1, 135.0)
        String def_d = "N";
        double def_lat = 35.1;
        double def_long = 135.0;

        String p_text;

        if (d_text.equals(def_d) && get_lat > def_lat
&& get_long == def_long){
            // 先行車の時、p_text に get_lat と get_long
の値を代入
            p_text = get_lat + ", " + get_long;
            // 取得したテキストを TextView に張り付
ける
            directionView.setText(d_text);
            positionView.setText(p_text);
            stateView.setText(s_text);

        }else if(! d_text.equals(def_d)){
            // N でないとき、"Not N!"を代入
            d_text = "Not N!";
            p_text = get_lat + "," + get_long;

            if (get_lat <= def_lat){
                // 前方にいる車でないとき、"Not
Forward!"を代入
                p_text = "Not Forward!";
            }
            // どちらかの条件が異なるとき、空白にする
            s_text = "";
            directionView.setText(d_text);
            positionView.setText(p_text);
            stateView.setText(s_text);

        }else{
            p_text = "Not Forward!";
            s_text = "";

            directionView.setText(d_text);
            positionView.setText(p_text);
            stateView.setText(s_text);
        }
    }
}
```

<activity_main.xml>

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/
apk/res/android"
        xmlns:app="http://schemas.android.com/apk/
res-auto"
        xmlns:tools="http://schemas.android.com/too
ls"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#dfe"
        tools:context=".MainActivity">

        <TextView
        android:id="@+id/direction_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:gravity="center_horizontal"
        android:textSize="30sp"

        app:layout_constraintEnd_toEndOf="parent
"

        app:layout_constraintHorizontal_bias="0.49
8"
```

```
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/position_text_view" />

            <TextView
            android:id="@+id/position_text_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal"
            android:textSize="30sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

            <TextView
            android:id="@+id/state_text_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="32dp"
            android:gravity="center_horizontal"
            android:textSize="30sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/direction_text_view" />

            <EditText
            android:id="@+id/latitude_Num"
            android:layout_width="66dp"
            android:layout_height="28dp"
            android:layout_marginTop="32dp"
            android:ems="10"
            android:autofillHints="@string/hint_latitude"
            android:background="#ffffff"
            android:hint="@string/hint_latitude"
            android:inputType="numberDecimal"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
            app:layout_constraintStart_toStartOf="parent"
```

```
            app:layout_constraintTop_toBottomOf="@+id
/state_text_view" />

            <EditText
            android:id="@+id/longitude_Num"
            android:layout_width="88dp"
            android:layout_height="30dp"
            android:layout_marginTop="32dp"

            android:autofillHints="@string/hint_longitud
e"
            android:background="#ffffff"
            android:ems="10"
            android:hint="@string/hint_longitude"
            android:inputType="numberDecimal"

            app:layout_constraintEnd_toEndOf="parent
"

            app:layout_constraintHorizontal_bias="0.49
8"

            app:layout_constraintStart_toStartOf="pare
nt"

            app:layout_constraintTop_toBottomOf="@+id
/latitude_Num" />

            <EditText
            android:id="@+id/direction_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="32dp"

            android:autofillHints="@string/hint_directio
n"
            android:background="#ffffff"
            android:hint="@string/hint_direction"
            android:inputType="text"

            app:layout_constraintEnd_toEndOf="parent
"

            app:layout_constraintHorizontal_bias="0.49
7"
            app:layout_constraintLeft_toLeftOf="parent"

            app:layout_constraintRight_toRightOf="pare
nt"

            app:layout_constraintStart_toStartOf="pare
nt"

            app:layout_constraintTop_toBottomOf="@+id
/longitude_Num" />

            <EditText
            android:id="@+id/state_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="32dp"
            android:autofillHints="@string/hint_state"
            android:background="#ffffff"
            android:hint="@string/hint_state"
            android:inputType="text"

            app:layout_constraintEnd_toEndOf="parent
"

            app:layout_constraintHorizontal_bias="0.49
8"
            app:layout_constraintLeft_toLeftOf="parent"

            app:layout_constraintRight_toRightOf="pare
nt"

            app:layout_constraintStart_toStartOf="pare
```

```
nt"

        app:layout_constraintTop_toBottomOf="@+id
/direction_text" />

        <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="@string/button"

        app:layout_constraintBottom_toBottomOf="
parent"

        app:layout_constraintEnd_toEndOf="parent
"
        app:layout_constraintLeft_toLeftOf="parent"

        app:layout_constraintRight_toRightOf="pare
nt"

        app:layout_constraintStart_toStartOf="pare
nt"

        app:layout_constraintTop_toBottomOf="@+id
/state_text" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

<string.xml>

```
<string.xml>

<resources>
        <string name="app_name">Relay Test
App</string>
        <string name="button">Button</string>
        <string
name="hint_latitude">Latitude</string>
        <string
name="hint_longitude">Longitude</string>
        <string
name="hint_direction">Direction</string>
        <string name="hint_state">State</string>
</resources>
```