平成２２年度　特別研究報告書

ＭＡＮＥＴを用いた高速道路における渋滞情報取得

龍谷大学　理工学部　情報メディア学科

学籍番号　　T070463　　松島　元気

指導教員　三好　力　教授

内容梗概

近年、無線通信技術の発展により、Mobile Ad-hoc Network(MANET)技術が注目されつつある。MANET の特徴として、ネットワークインフラをほぼ必要としない、マルチホップ通信でネットワークを構築できるという利点がある。インフラをほぼ必要としないという点に着目して、現在膨大なインフラ費用がかかっている日本の渋滞情報取得システム VICS を、MANET を用いて改善しようと試みた。

提案手法１については、性能実験結果から、正常にシステムが稼動していることが確認できたが、提案手法２に比べるとシステムに対しての負荷が多少大きい。設備費用においては、評価実験により、既存手法に比べ１/4 から 1/8 削減できると確認できた。提案手法２についても性能実験結果から、正常にシステムが稼動していることが確認できた。設備費用においては、評価実験により、既存手法に比べ１/2 から 1/4 削減できると確認できた。

本研究で、MANET を用いた渋滞情報取得システムにより、既存手法に比べ、今回提案した手法によって設備費用投資には大幅な削減が可能だと検証できた。

目次

第１章　はじめに

1.1　本研究の背景

各家庭にある自動車においては、情報端末であるカーナビゲーションシステムが搭載される割合が非常に高くなってきている。マイボイスコムは 2008 年 12 月 22 日までに、カーナビ(カーナビゲーションシステム)に関する調査結果を発表した。それによると、調査母体の中でカーナビ付の自動車を所有しているは全体の 5 割近くに登っていたことが明らかになった。

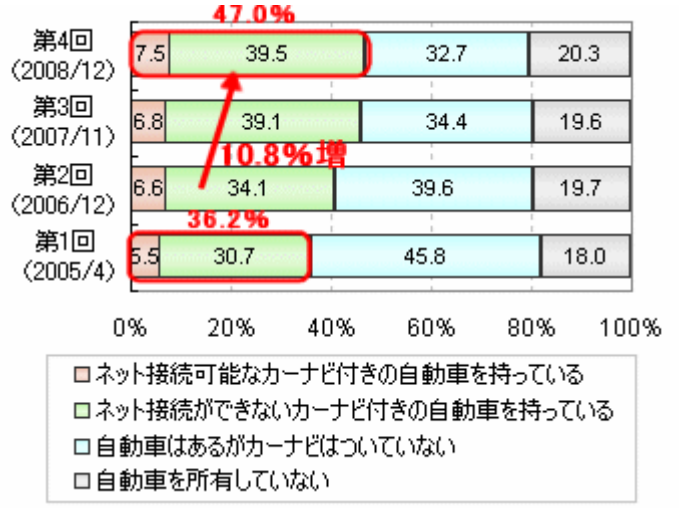

図１　カーナビ普及率

道に迷っても大型サイズの地図に頼る必要もなく、目的地を指定すれば適切な道案内もしてくれる、登場以前の人が見たら魔法の道具にすら見えるかもしれない 「カーナビ」ことカーナビゲーションシステム。当初はデータを CD などの外部記憶媒体で提供するだけだったが、最近ではネットに接続して多種多様なサービスを提供する機能もかね備わったものも登場している。

現在、日本をはじめとする世界各国において、交通渋滞は問題になっている。国土交通省によると、渋滞による損失は日本全国で年間約 38 億時間、金額にして約 12 兆円と試算されている。引用[１]

その交通渋滞を緩和するため、日本ではＶＩＣＳや各カーナビゲーションシステムベンダー独自の渋滞情報システムが導入されている。以下では、これらの既存技術を解説する。

1.1.2　VICS

VICS は、ドライバーのニーズに即し利便性を向上させるとともに、輸送時間の短縮によるコストの削減、的確な状況把握による安全性の向上、交通の円滑化による環境の保全等を可能とし、ひいてはゆとりある国民生活の実現と社会経済の発展に寄与することを目的としている。

高速道路や一般道路での渋滞をなくし、交通事故を減らし、道路環境を改善することは、現在世界各国が抱える共通のテーマとなっている。こうした状況の中、ドライバーの適正なルート選択を促し、快適でスムーズなドライブをサポートする VICS は、交通流を適切に分散させ、道路交通の安全性や円滑性を向上 させ、さらには道路環境を改善するシステムとして活躍している。

VICS は、VICS センターで編集、処理された渋滞や交通規制などの道路交通情報をリアルタイムに送信し、カーナビゲーションなどの車載機に文字・図形で表示させています。

VICS は３つの通信・放送のメディアで情報を提供しています。１つ目は、情報収集。道路交通情報を体系的に収集する（図２）。２つ目は情報処理・編集。収集された情報を処理

図２　情報の流れ

編集する。VICS センターでは、収集された情報を 3 つのメディアを通じて、24 時間 365 日、全国で提供している（但し、休止日を除く）。収集された情報はリアルタイムでカーナビに送信。その際に、3 つの表示タイプで見やすく表示できるように、データの処理・編集を行っている。３つ目は、情報提供。処理された情報をカーナビに伝送しる。引用[2]

FM 多重放送（NHK 等の各地の FM 放送局）

各地の NHK・FM 放送局から当該都道府県とその周辺の道路交通情報を提供します。

■渋滞情報や所要時間情報
■事故や故障車、工事、災害、気象条件等による規制情報
■駐車場の位置と満車・空車情報

電波ビーコン（高速道路）

主に高速道路に設置され、概ね自車の前方 200km 以内の道路交通情報を提供する。

■インターチェンジ間の所要時間
■渋滞情報や分岐案内（平行する主要一般道含む）
■事故や故障車、工事、災害、気象条件等による規制情報
（通行止、車線規制、速度規制、チェーン規制等）

光ビーコン（一般道の主要幹線道路）

主要な一般道路に設置され、概ね自車の前方 30km 以内と後方 1km 以内の道路交通情報を提供する。

■渋滞情報や所要時間情報
■事故や故障車、工事、災害、気象条件等による規制情報
（通行止、車線規制、速度規制、チェーン規制等）
■駐車場の満車・空車情報

1.1.3　HONDA インターナビシステム

VICS より更に進んだシステムとして、ホンダ（本田技研工業）は、HONDA インターナビシステムを導入している[3]。これは、同社のカーナビゲーションシステムユーザーを対象とした独自のテレマティクスを中心とした交通情報サービスである。携帯電話接続によるオンデマンド型の交通情報サービス「インターナビ VICS」を最大の特徴とするが、無料の会員登録制の「インターナビ・プレミアムクラブ」、地図の無償更新サービスやインターネットの WEB サイトと連携したパーソナルホームページおよび有料オプションでのロードアシスタンスサービス「QQ コール」などをあわせて提供する。インターナビ交通情報は、全国のＶＩＣＳ情報＋フローディングカーデータからなる。「全国の VICS 情報」と「フローティングカーデータ」をインターナビ情報センターで統合。普通のカーナビでは得ることのできない豊富な情報が詰まった Honda 独自の「インターナビ交通情報」をつくり上げている。それを、ユーザーがデータ通信によって受け取り、ルート案内などに活かす独自のネットワークを形成している。

フローティングカーシステムとは、日本中のインターナビ装着車の走行データ（フローティングカーデータ）を集め、ルート案内などに活かすシステムです。Honda が世界で初めて実用化した。 高速道路や主要幹線道路のみを対象としている VICS 情報と違い、インターナビ装着車が実際に走ったデータなので細かい道も対象となり、ルート案内の幅を広げる（図３）。

VICS が対象としない細かい道の情報まで集まるのが、フローティングカーシステムのひとつの特徴である。高速道路の渋滞を避けたのに、一般道も大渋滞。そんなストレスから解放。これは、一般道が VICS 対象道路でない場合によくあるケースである。インターナビなら、フローティングカーシステム によって一般道の渋滞も把握しているため、もしフローティングカーシステム対象道路を利用したルートが空いていれば、多少遠回りでもそちらを案内することができる。高速道路と一般道が同じように渋滞している場合は、早く到着できるルートの方を案内する（図４）。

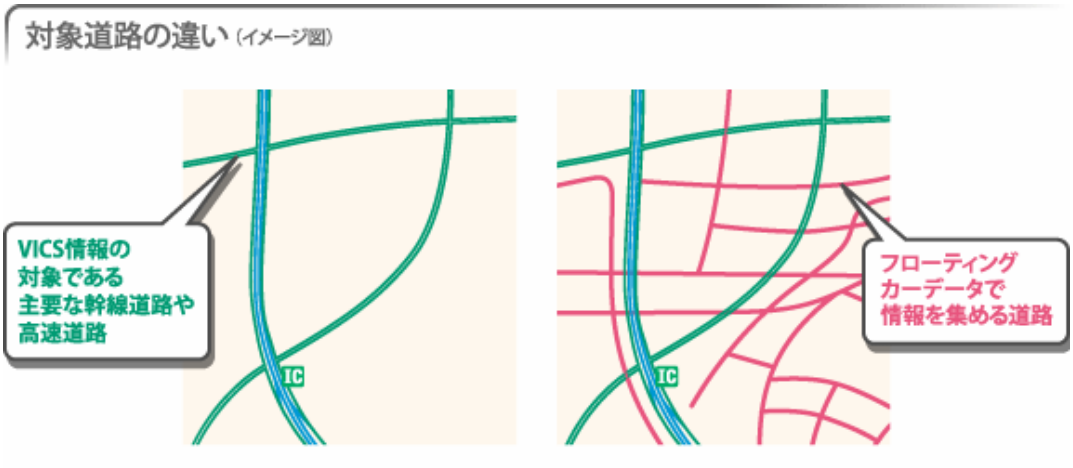

図３　フローティングカーシステムによる対象道路の違い

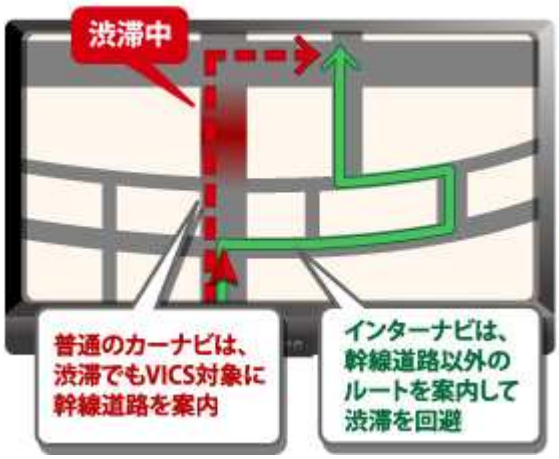

図４　インターナビの経路案内

しかし、そんなＨＯＮＤＡインターナビシステムにも欠点はある。少しの渋滞でも、高速道路のの途中で、高速道路から下ろして一般道を走らせたりする。しかし、１０分ぐらいの到着時間が遅くなるぐらいならば、高速をそのまま使った方が料金面で、得だったりする。実際、ナビの経路を無視して、そのまま高速を走ると、渋滞が解消していたり、混雑しているが 50km/h で走れたりする例が多数報告されている。また、インターナビシステムは、インターネットと接続するために、携帯電話などの通信機器が必要とされる。

1.1.4　車車間通信

車車間通信とは、四輪車や二輪車が無線通信で互いに情報交換する技術である。既存技術としては、日本では本田技研工業が、2008 年に実用化した技術である。本田技研工業の場合、開発運転席から見えにくい場所にいるほかの車の存在を運転者に知らせて、事故回避を促すための仕組みである。車どうしの通信は、200 メートル以内にいる最大 120 台が同時に情報を交換できる。近くにいる車の位置、走っている方向、速度などをカーナビの画面に表示する。事故を引き起こすかもしれない車両が近づいた場合、四輪車ではアクセルペダルを振動させたり、シートベルトを自動的に軽く締めたりして注意を促す。二輪車では、ヘルメットに内蔵されたスピーカーを通じて音声で危険を知らせる。車どうしの情報交換は各社が実用化を目指していて、通信規格の統一化も協議されている。

1.2　本研究の目的

ＶＩＣＳには２つの弱点がある。１つ目は道路交通情報を一度センターに集め処理編集を経て配信が行われ、更新間隔は最速で 5 分だが、FM 電波の感度が悪い地域やビーコンがない道を走っている間は、受信ができないため、タイムラグが起こってしまう。２つ目はＶＩＣＳの情報収集インフラ、発信インフラに莫大な金額がかかることである。ＭＡＮＥＴ車車間通信を用いた渋滞情報管理システムは、インフラなしで、１台対１台の車車間通信を繰り返して、車の台数、速度等情報を交換することで渋滞の情報を集めるというものである。この研究によって、インフラを用いないため、設備費のコスト削減が期待できる。本研究の目的は、ＶＩＣＳのようにインフラを用いることなく、各車両が備える情報端末による無線機通信等を介して、ＭＡＮＥＴを用いた車車間通信を行い、渋滞情報の把握を行うことである。

1.3　本論文の流れ

本論文は以下の章により構成される。

第１章では、本研究の概要について述べる・

第２章では、Mobile Ad-hoc Network(MANET)の概要について述べる。

第３章では、提案手法、実験環境と実験内容、及びに実験結果を示し、それに対する考察を行う。

第４章では、本論文に関するまとめと今後の課題について述べる。

第２章　Mobile Ad-hoc Network

本章では Mobile Ad-hoc Network (MANET) の概要について述べる。

2.1　MANET とは

MANET とは、従来のインターネットと違いイーサネットや無線 LAN のアクセス・ポイントといった基地局などのインフラを必要とせず、自律的に形成できるネットワークのことを指し示す。つまり、端末同士が直接接続して構成するネットワークのことである。事前にインフラを用意しなくても、まさにその場でネットワークを作り上げることができる。無線通信を使ってデータを端末から端末へバケツ・リレーのように手渡すことから、「マルチホップ・ネットワーク」とも呼ぶ。無線 LAN の普及に伴って注目を集めている。従来のインターネットを図　で示す。今日、ＭＡＮＥＴの技術はニンテンドーＤＳやＰＳＰなどの携帯ゲーム機でも使われている。

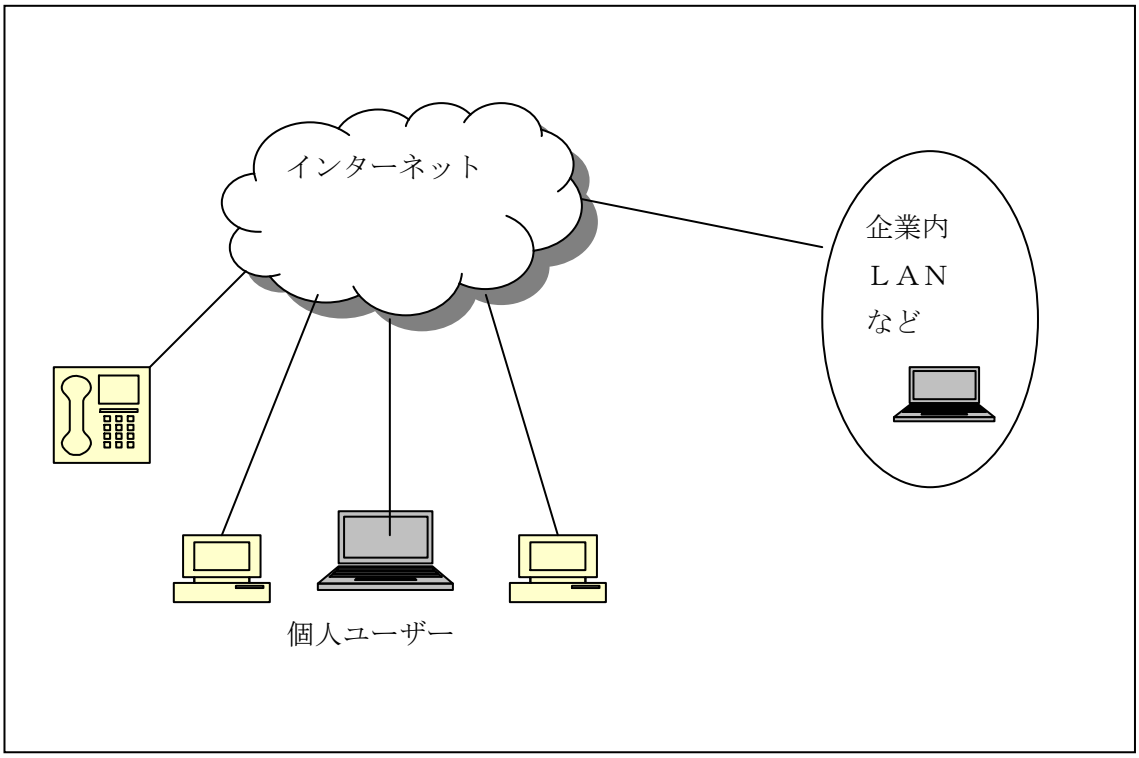

図５　既存のインターネット

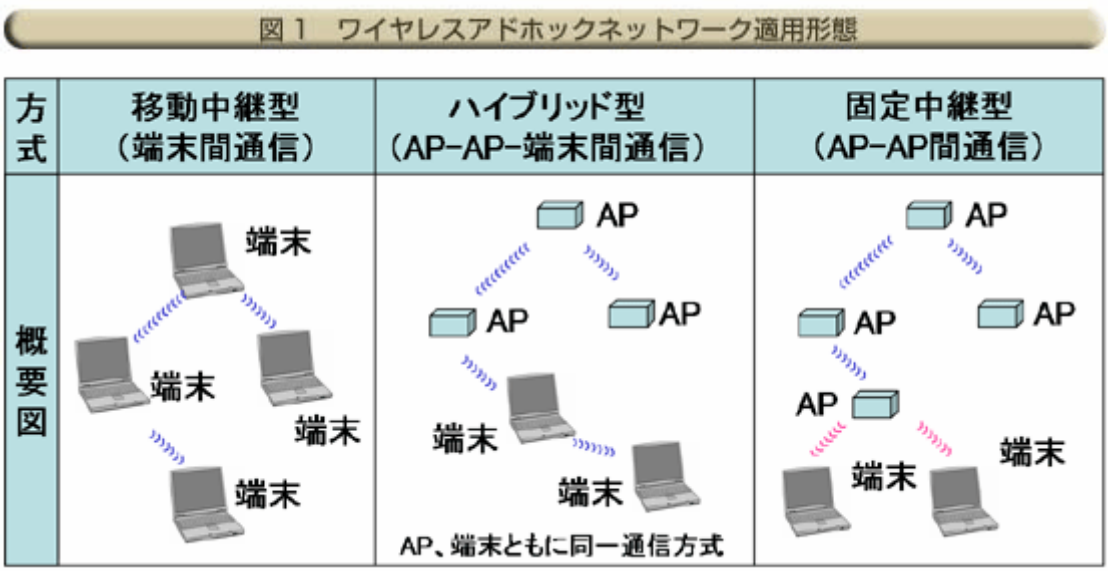

図６　ＭＡＮＥＴの仕組み[引用４]

ワイヤレスアドホックネットワークは自律分散型であり、移動中継型、ハイブリッド型、固定中継型の 3 種のネットワークの運用形態がある（図６）。

NTT では少ない工事稼動でネットワーク構築が可能で、かつ安定なサービス提供が可能なアクセスポイント（AP）間をワイヤレスアドホックネットワークで構築する固定中継型あるいはハイブリッド型での運用を考えており、これらの高スループット化、 VoIP・映像系アプリに対応するための QoS 制御、ハンドオーバ技術、またセキュリティ確保などの研究開発を行っている。

(1)高スループット化

単一 Ch でのワイヤレスアドホックネットワーク構築に無線 LAN（IEEE802.11）を使用する場合、隠れ端末問題、さらし端末問題によってスルー プットが低下する。このため、複数 Ch 運用が有効で、自律的な Ch 割当、ルーティング、効率的なパケット転送方法が課題となる。

(2)QoS 制御

VoIP や映像情報送信などの利用が考えられ、ワイヤレスアドホックネットワークにおいても、優先制御・帯域制御技術などの QoS 制御が課題となる。

(3)ハンドオーバ

VoIP や映像情報送信などの利用シーンとして歩行程度の速度での移動を考慮した場合に、アクセスポイントをまたがって移動することも考えられることから、可能な限り品質劣化を抑えるためのハンドオーバ技術が必要となる。

(4)セキュリティ確保

ワイヤレスネットワークでは盗聴防止、端末のなりすまし防止、パケット改竄防止などのセキュリティ確保が必須であり、アドホックネットワークでは、さら に不正なアクセスポイントの設置が容易であること、また認証パケットなどを模擬した不正パケットの大量侵入により無線帯域が大幅に消費され通信が妨害される可能性がある。このため、アクセスポイント自身の認証、不正パケットの検出・転送防止策も重要となる。引用[5]

2.2　MANET にはどんな未来があるか

広帯域無線システムを使用した無線アクセスサービスのエリア拡大を経済的かつ効率的に図ることができるため、ユーザにとっては利便性向上による値ごろ感を感じられるサービス提供を図ることができる。また、災害時における迅速な通信ネットワークの構築が可能であり、被災地における状況把握、情報提供、連絡手段の確保が可能となる（図７）

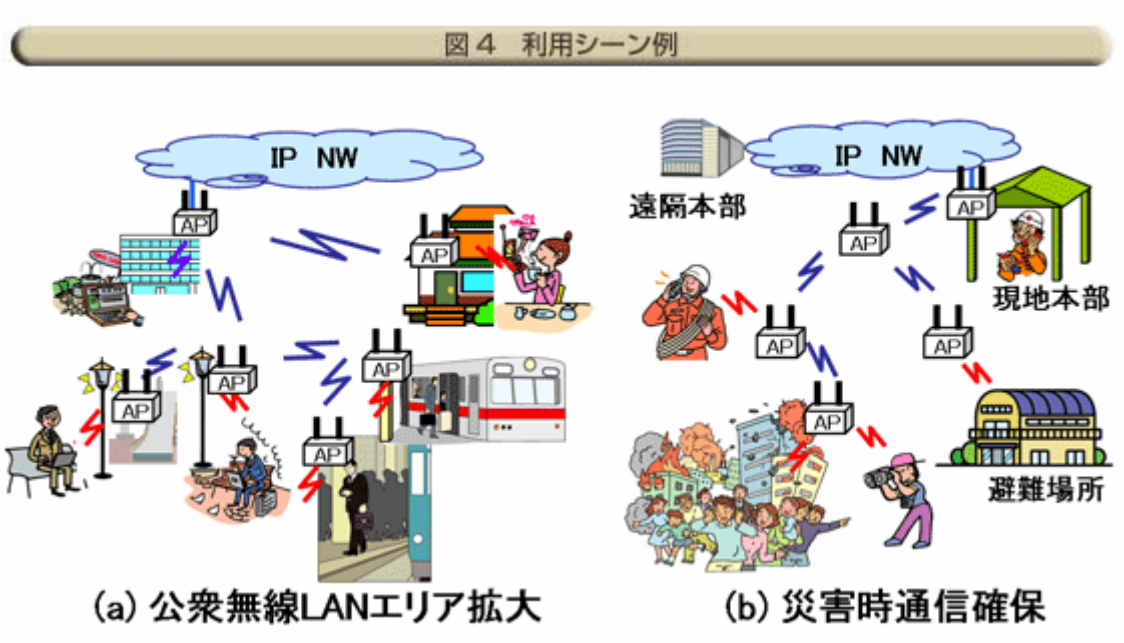

図７　　利用シーン例

第３章　MANET 車車間通信を用いた高速道路における渋滞情報管理システムと実験及び評価

本章では、1.2 節で述べた問題点に焦点をおき、新手法を提案する。

3.1 概要

ここで提案する手法は、既存のＶＩＣＳやＨＯＮＤＡインターナビシステムの問題点をなくすものとする。問題点とは 1.2 節で示したもので、渋滞情報の取得をする手法を提案する。利用範囲は高速道路に限定している。なぜならば、高速道路は道が一本で交差点などがなく、上りと下りしか進行方向がないため、上りと下りのすれ違い通信で必ず情報のやりとりができるからである。

3.2 提案手法１

提案手法１は、高速道路において、ＶＩＣＳなどの設備インフラを用いず、車車間同士で通信をして渋滞情報を得るものである。1.2 節で示した問題点を解決するために行う。適用範囲を高速道路のみに限定するのは、一般道と違って右折左折がなく、道路が対面通行であるため確実に自分の対向車からＸｋｍ先の情報を入手できるためである。カーナビゲーションは以下の機能を持つものと仮定してシミュレーションを行う。

--IEEE802.11x 規格の無線ＬＡＮ装置
--GPS
--数十ＧＢ程度のＨＤＤまたはＳＳＤ
--地図データを納めたメディアとドライブ

高速道路を走行する車の進行方向、上りと下りはＧＰＳを用いて進行方向を得る。ＧＰＳは、自車位置を得るためにも使う。シミュレーションプログラムでは、自車の位置を決めて、ＧＰＳを使ったと仮定する。速度、位置、方向を車にもたす。時間がたつと走行速度から割り出して位置がかわる。

また、実際に高速道路上を車が走るシミュレータを作成した。シミュレーションは、高速道路の渋滞情報を調べるために用いる。それぞれの車が持っている情報（情報を交換した車の台数、速度、位置、方向）を用いる。渋滞情報を得るためには、自車の進行方向のＸｋｍ先の情報が必要である。そのために、距離地点別 10km 毎における平均速度で渋滞しているかしていないかを判定する。

3.2.1 アルゴリズム詳細（提案手法１）

提案手法１について以下にシミュレーション実験を行った際に用いたアルゴリズムを説明する。

・上りの車線の車についての処理
  ・シミュレータ上の車の台数を乱数で決める
  ・各車１台１台の時速を乱数で求める
  ・各車１台１台の分速を乱数で求める
  ・各車１台１台の 10 分ごとの現在位置を求める
  ・各車の現在位置を１の位で四捨五入し、現在位置を 10km 単位にする
  ・すれ違った対向車線の車から各地点における平均速度を受け取る

・下りの車線の車についての処理
  ・シミュレータ上の車の台数を乱数で決める
  ・各車１台１台の時速を乱数で求める
  ・各車１台１台の分速を乱数で求める
  ・各車１台１台の 10 分ごとの現在位置を求める
  ・各車の現在位置を１の位で四捨五入し、現在位置を 10km 単位にする
  ・すれ違った対向車線の車から各地点における平均速度を受け取る

・上り車線についての渋滞情報の処理
  ・平均速度に応じて、渋滞情報を出力
    ・もし、平均速度が２０ｋｍ以下なら混雑レベル[大]
    ・もし、平均速度が４０ｋｍ以下なら混雑レベル[中]
    ・もし、平均速度が６０ｋｍ以下なら混雑レベル[小]
    ・もし、平均速度が８０ｋｍ以下なら混雑レベル[ほぼ混雑なし]
    ・もし、平均速度が８０ｋｍ以上なら混雑レベル[混雑なし]

・下り車線についての渋滞情報の処理
    ・平均速度に応じて、渋滞情報を出力
    ・もし、平均速度が２０ｋｍ以下なら混雑レベル[大]
    ・もし、平均速度が４０ｋｍ以下なら混雑レベル[中]

・もし、平均速度が６０ｋｍ以下なら混雑レベル[小]
・もし、平均速度が８０ｋｍ以下なら混雑レベル[ほぼ混雑なし]
・もし、平均速度が８０ｋｍ以上なら混雑レベル[混雑なし]

3.2.2　実験環境　(提案手法１提案手法２共通部)

提案手法１､提案手法２の共通の実験環境は､対面通行の総延長 200ｋｍの高速道路とし、高速道路上には、乱数で発生させたランダムの数の車が走行している

・実際に車が走るシミュレータを作成した。シミュレータの環境を以下に示す。
・直線 200km の高速道路区間（図８）
・その区間は制限速度 100km とする。
・高速道路上を走る車の速度は乱数で割り出す。

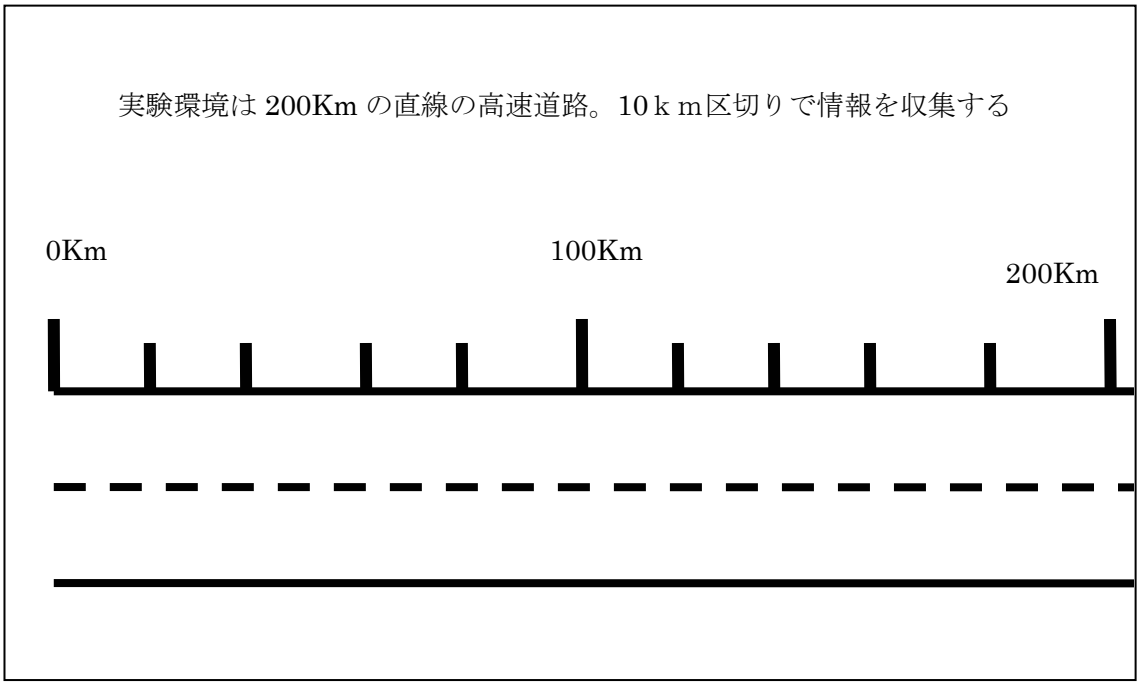

図８　シミュレーション環境

3.2.3　実験環境（提案手法１）

提案手法１のみに用いる実験環境を以下に示す。

・200km の道路を 10km 間隔で区切り、その区切った地点ですれちがい通信（車車間通信）を行う。（図９）

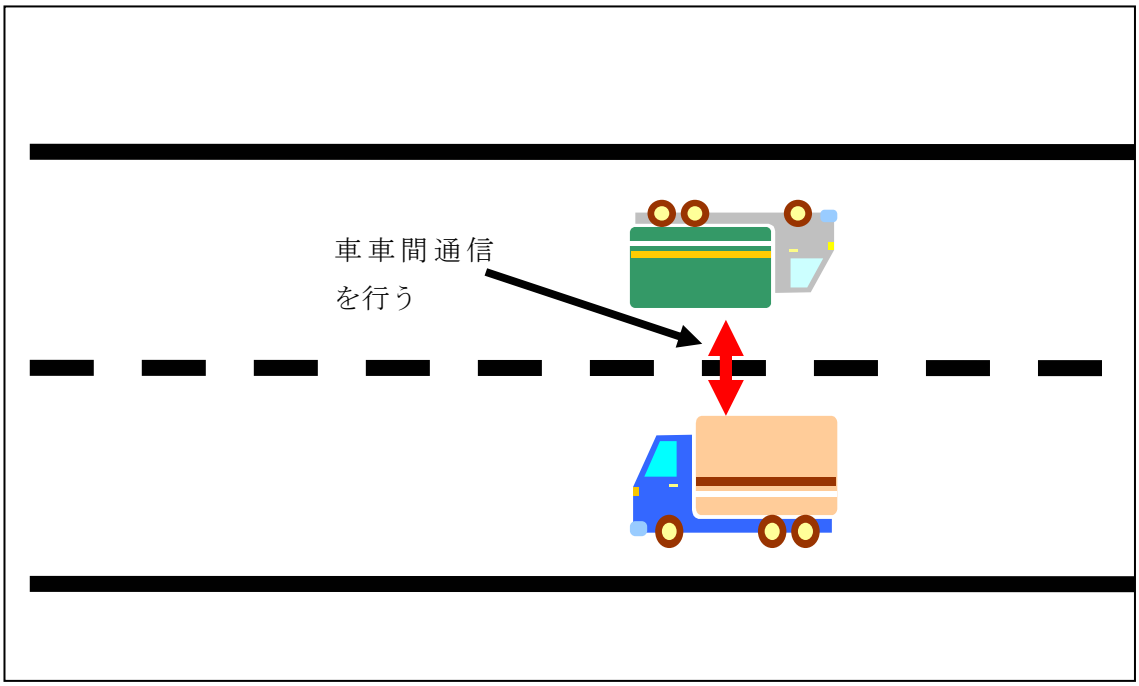

図９　シミュレーション環境

3.2.4　システムの性能検証（性能実験）

この節では作り上げたシミュレーションシステム（提案手法１）の性能の検証を行う。最初にこのシミュレータシステム上で、意図的に全領域で渋滞しない状況を作り出した場合の実行結果を図１０に示す。

```
上りの車線の車
1 台目の 10 分後の車の位置は、13km 地点で
す
10km 地点の情報を出力　==混雑レベル[混雑なし]==
20km 地点の情報を出力　==混雑レベル[混雑なし]==
30km 地点の情報を出力　==混雑レベル[混雑なし]==
40km 地点の情報を出力　==混雑レベル[混雑なし]==
50km 地点の情報を出力　==混雑レベル[混雑なし]==
60km 地点の情報を出力　==混雑レベル[混雑なし]==
```

```
70km 地点の情報を出力　==混雑レベル[混雑なし]==
80km 地点の情報を出力　==混雑レベル[混雑なし]==
90km 地点の情報を出力　==混雑レベル[混雑なし]==
110km 地点の情報を出力　==混雑レベル[混雑なし]==
120km 地点の情報を出力　==混雑レベル[混雑なし]==
130km 地点の情報を出力　==混雑レベル[混雑なし]==
140km 地点の情報を出力　==混雑レベル[混雑なし]==
150km 地点の情報を出力　==混雑レベル[混雑なし]==
160km 地点の情報を出力　==混雑レベル[混雑なし]==
170km 地点の情報を出力　==混雑レベル[混雑なし]==
180km 地点の情報を出力　==混雑レベル[混雑なし]==
190km 地点の情報を出力　==混雑レベル[混雑なし]==
200km 地点の情報を出力　==混雑レベル[混雑なし]==
```

図１０　　実行結果（一部）

実行結果を見て分かる通り、渋滞していない状況をデータで入力し実行すると、全領域で渋滞していないという結果が得られている。

また、反対に意図的に渋滞を作り出して検証する。30km、60km、200km 地点で意図的に渋滞するようデータを入力して実行すると図１１のようになる。

```
上りの車線の車
1 台目の 10 分後の車の位置は、8km 地点で
す
10km 地点の情報を出力　==混雑レベル[混雑なし]==
20km 地点の情報を出力　==混雑レベル[混雑なし]==
30km 地点の情報を出力　==混雑レベル[大]==
40km 地点の情報を出力　==混雑レベル[混雑なし]==
50km 地点の情報を出力　==混雑レベル[混雑なし]==
60km 地点の情報を出力　==混雑レベル[大]==
70km 地点の情報を出力　==混雑レベル[混雑なし]==
80km 地点の情報を出力　==混雑レベル[混雑なし]==
90km 地点の情報を出力　==混雑レベル[混雑なし]==
100km 地点の情報を出力　==混雑レベル[混雑なし]==
110km 地点の情報を出力　==混雑レベル[混雑なし]==
120km 地点の情報を出力　==混雑レベル[混雑なし]==
130km 地点の情報を出力　==混雑レベル[混雑なし]==
```

```
140km 地点の情報を出力　==混雑レベル[混雑なし]==
150km 地点の情報を出力　==混雑レベル[混雑なし]==
160km 地点の情報を出力　==混雑レベル[混雑なし]==
170km 地点の情報を出力　==混雑レベル[混雑なし]==
180km 地点の情報を出力　==混雑レベル[混雑なし]==
190km 地点の情報を出力　==混雑レベル[混雑なし]==
200km 地点の情報を出力　==混雑レベル[大]==
```

図１１　実行結果（一部）

両者を比較してわかるように、このシシミュレーションステムは正常に動作していることが検証できた。

3.3 提案手法 2

提案手法１では、移動中継型のアドホックネットワークの形態を用いていたが、提案手法２では、ハイブリッド型（ＡＰ－ＡＰ－端末間通信）の形態を用いる。提案手法１では、10km 地点ごとに端末間で通信を行っていたが、今回は 10km ごとにＡＰ（アクセスポイント）を設けて、そこに情報を集める。ＡＰ同士はマルチホップ通信を行い、バケツリレー方式で 0km 地点の AP から 200km 地点まで AP の渋滞情報を交換し合い、全領域の渋滞情報を把握する。

3.3.1 アルゴリズム詳細(提案手法 2)

以下にシミュレーション実験を行った際に用いたアルゴリズムを説明する。
提案手法２について以下にアルゴリズムを示す。

・上りの車線の車についての処理
　・シミュレータ上の車の台数を乱数で決める
　・各車１台１台の時速を乱数で求める
　・各車１台１台の分速を乱数で求める
　・各車１台１台の 10 分ごとの現在位置を求める
　・各車の現在位置を１の位で四捨五入し、現在位置を 10km 単位にする
　・Xkm 地点ごとにあるアクセスポイントに各車は現在位置と走行速度情報を送る
　・各アクセスポイントは各車の現在位置と走行速度と車の台数情報を蓄える

・下りの車線の車についての処理
  ・シミュレータ上の車の台数を乱数で決める
  ・各車１台１台の時速を乱数で求める
  ・各車１台１台の分速を乱数で求める
  ・各車１台１台の 10 分ごとの現在位置を求める
  ・各車の現在位置を１の位で四捨五入し、現在位置を 10km 単位にする
  ・Xkm 地点ごとにあるアクセスポイントに各車は現在位置と走行速度情報を送る
  ・各アクセスポイントは各車の現在位置と走行速度と車の台数情報を蓄える

・上り車線についての渋滞情報の処理
  ・各地点における平均速度を求める。求める式は
      平均速度　＝１台１台の車速の合計　÷　車の台数
  ・平均速度に応じて、渋滞情報を出力
    ・もし、平均速度が２０ｋｍ以下なら混雑レベル[大]
    ・もし、平均速度が４０ｋｍ以下なら混雑レベル[中]
    ・もし、平均速度が６０ｋｍ以下なら混雑レベル[小]
    ・もし、平均速度が８０ｋｍ以下なら混雑レベル[ほぼ混雑なし]
    ・もし、平均速度が８０ｋｍ以上なら混雑レベル[混雑なし]

・下り車線についての渋滞情報の処理
  ・各地点における平均速度を求める。求める式は
      平均速度　＝１台１台の車速の合計　÷　車の台数
  ・平均速度に応じて、渋滞情報を出力
    ・もし、平均速度が２０ｋｍ以下なら混雑レベル[大]
    ・もし、平均速度が４０ｋｍ以下なら混雑レベル[中]
    ・もし、平均速度が６０ｋｍ以下なら混雑レベル[小]
    ・もし、平均速度が８０ｋｍ以下なら混雑レベル[ほぼ混雑なし]
    ・もし、平均速度が８０ｋｍ以上なら混雑レベル[混雑なし]

3.3.2　実験環境（提案手法 2）

提案手法２のみに用いる実験環境を以下に示す。

・200km の道路を 10km 間隔で区切り、その区切った地点に AP（アクセスポイント）

を設置する。AP と各車間で通信を行い、情報を集める。（図 12）

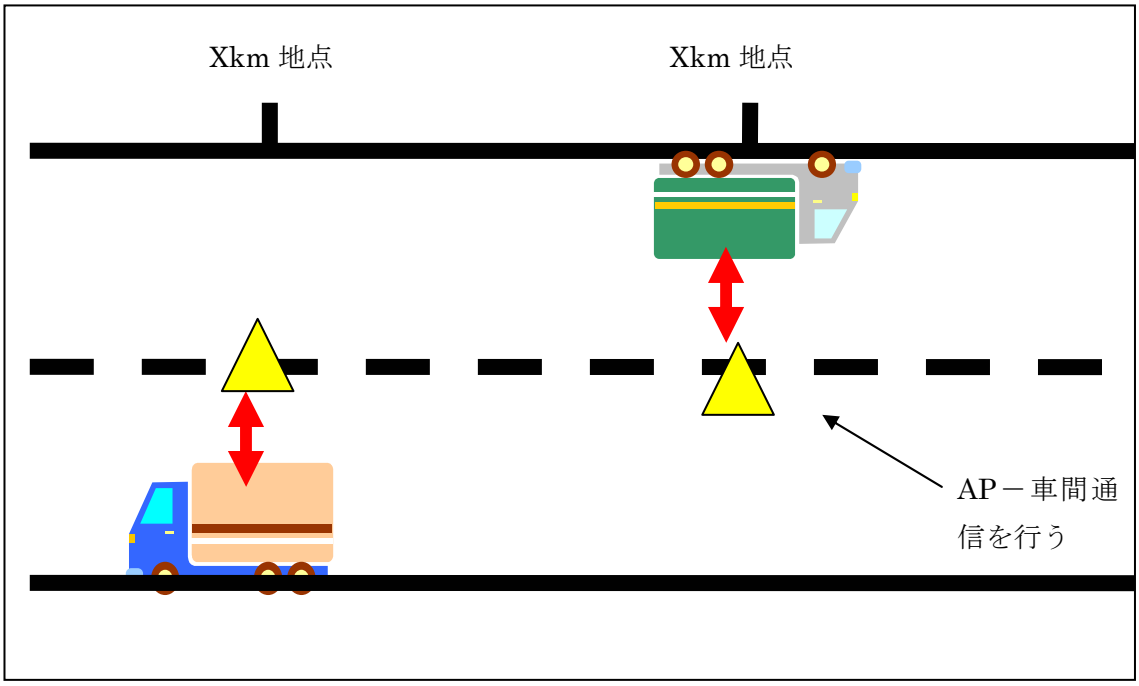

図 12　シミュレーション環境

3.3.3　システムの性能検証（性能実験）

この節では作り上げたシミュレーションシステムの性能の検証を行う。
最初にこのシミュレータシステム上で、意図的に全領域で渋滞しない状況を作り出した場合の実行結果を図 13 に示す。

```
==============上り車線渋滞情報処理===============
10km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
20km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
30km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
40km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
50km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
60km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
70km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
```

```
80km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
90km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
100km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
110km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
120km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
130km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
140km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
150km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
160km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
170km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
180km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
190km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
200km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
================================================
```

図 13　　実行結果（一部）

実行結果を見て分かる通り、渋滞していない状況をデータで入力し実行すると、全領域で渋滞していないという情報がしっかりと得られている。

また、反対に意図的に渋滞を作り出して検証する。30km、60km、200km 地点で意図的に渋滞するようデータを入力して実行すると図１４のようになる。

```
==============上り車線渋滞情報処理===============
10km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
20km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
30km 地点の渋滞情報を出力　==混雑レベル[中]==
40km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
50km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
60km 地点の渋滞情報を出力　==混雑レベル[小]==
70km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
80km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
90km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
100km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
110km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
```

```
120km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
130km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
140km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
150km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
160km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
170km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
180km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
190km 地点の渋滞情報を出力　==混雑レベル[混雑なし]==
200km 地点の渋滞情報を出力　==混雑レベル[大]==
================================================
```

図１４　実行結果（一部）

両者を比較してわかるように、このシシミュレーションステムは正常に動作していることが検証できた。

3.4　システム実行速度の比較（性能実験）

提案手法１と提案手法２での、システムの実行速度を比較した。図１５に示す。

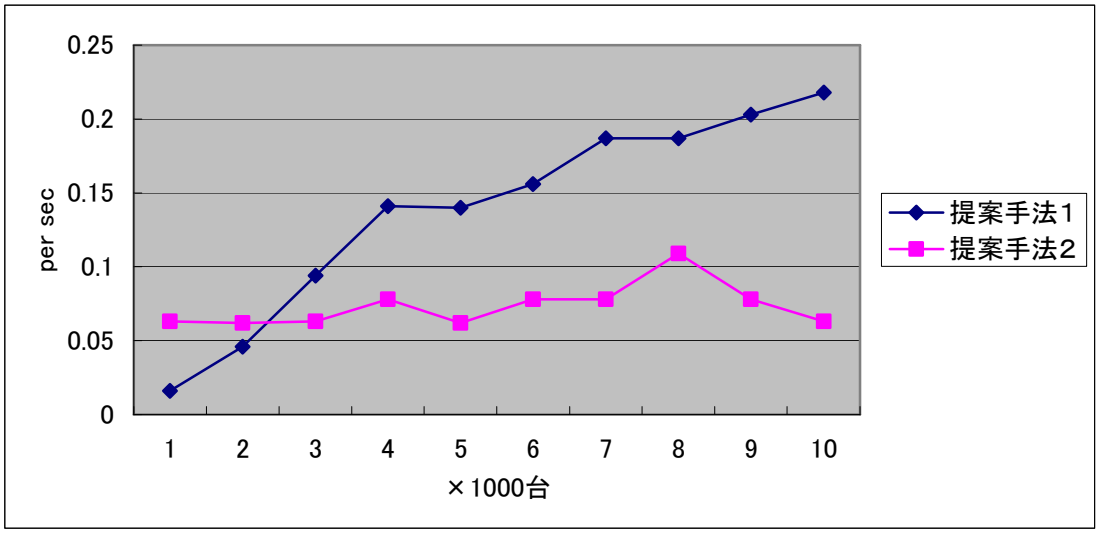

図１５　システム実行速度の比較

実験に使用した計算機のスペックは、CPU が Intel Core 2 Duo processor 1.4Ghz で、メモリが 4GB のものである。提案手法１では、車の台数に比例して実行速度が遅くなるが、提案手法２では車の台数が増えても実行速度にほぼ変化がないことがわかった。

3.5　設備費用の検証（評価実験）

この節では、既存手法と提案手法の、設備費用の検証を行う。VICS は高速道路においては、電波ビーコンを設置している。設置間隔は２～４km ごとである。電波ビーコンは１基約 400 万円の費用がかかる。引用[5][6]それにたいして、アドホック通信に使う IEEE802.11x 基準の無線 LAN 装置は平均価格は 4000 円程度で、提案手法２に用いる長距離無線 LAN は一基約２５万円[8]である。検証は以下の条件で行う。

- 直線 200km における設備費用を求める
- 電波ビーコンが 2km 間隔に設置してある場合の費用（既存手法１）
- 電波ビーコンが 4km 間隔に設置してある場合の費用（既存手法２）
- 提案手法１での費用（インフラは車自身、車は１万台走行しているものとする）
- 提案手法２での費用（10km ごとにアクセスポイントが１個ある）
- 本体のみの価格で、設置費用は含まない

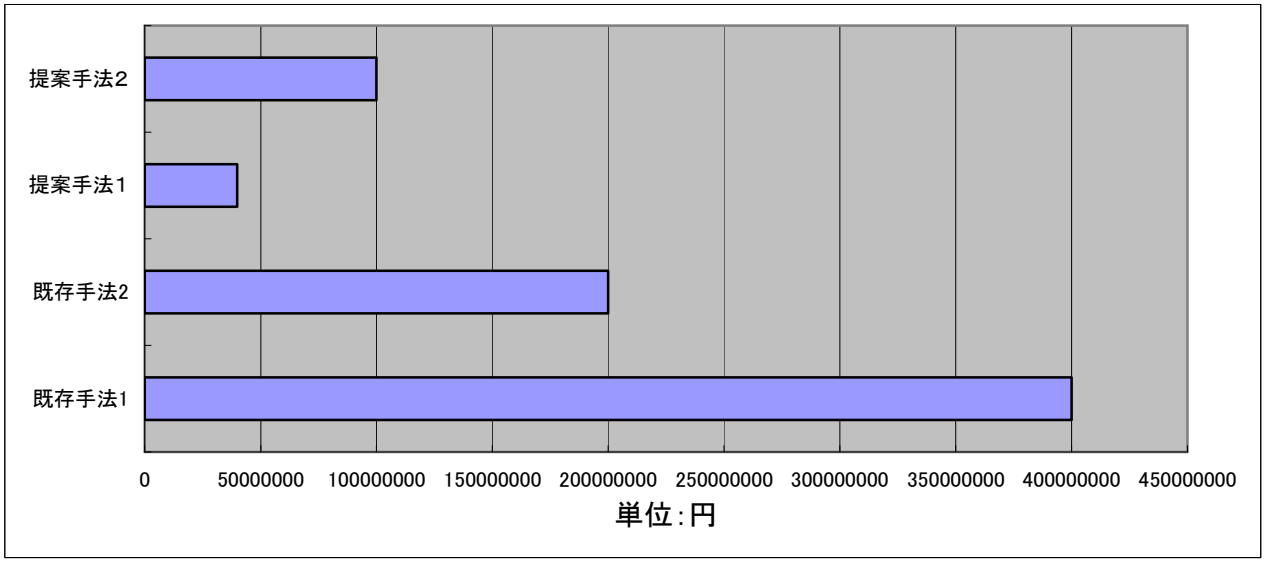

図１６　既存手法と提案手法での費用の比較

既存手法１だと設備費用は約４億円、既存手法２だと約２億円、提案手法１だと４０００万円（インフラは車１万台と仮定)、提案手法２だと約１億円になることがわかった。よって、既存手法に比べ、今回提案した手法によって設備費用投資には大幅な削減が可能だと検証できた。

第４章　おわりに

本論文では、既存の渋滞情報取得システムの弱点である莫大な設備費用を軽減できることができる MANET を用いた渋滞情報取得システムを提案した。

提案手法１については、性能実験結果から、正常にシステムが稼動していることが確認できたが、提案手法２に比べるとシステムに対しての負荷が多少大きい。設備費用においては、評価実験により、既存手法に比べ１/4 から 1/8 削減できると確認できた。提案手法２についても性能実験結果から、正常にシステムが稼動していることが確認できた。設備費用においては、評価実験により、既存手法に比べ１/2 から 1/4 削減できると確認できた。

本研究で、MANET を用いた渋滞情報取得システムにより、既存手法に比べ、今回提案した手法によって設備費用投資には大幅な削減が可能だと検証できた。また、性能実験から、本システムは正常に動作していることが示され、本研究の有効性を確認できた。

既存手法の VICS の２つの弱点であるうちの、膨大な設備費用の削減の有効性は確認されたが、もう１つの弱点であるタイムラグの発生についても考慮した手法も構想したい。

謝辞

本研究を進めるにあたり、指導教諭である龍谷大学理工学部情報メディア学科の三好力教授には、常日頃から多忙な際も研究についての貴重なアドバイスを頂いたばかりでなく、生活面においても大変配慮していただき厚く御礼申し上げます。

また、研究室の皆様にも公私にわたり様々なアドバイスをしていただき多くのことを学ばせていただきました。ここに深く感謝を申し上げます。

最後に、これまで支えてくれた家族、友人たちに心より感謝申し上げます。

参考文献

[1].http://www.gamenews.ne.jp/archives/2009/01/41_15.html
[2]. http://www.vics.or.jp/structure/index.html
[3]. http://www.honda.co.jp/internavi/
[4]. http://www.ntt.co.jp/mirai/organization/organization0305.html
[5]. http://www.web-pbi.com/its/old/its.htm
[6]. http://home.jeita.or.jp/dha/term/yougo-answ.html
[7]. 久保田　周治,802.11 高速無線 LAN 教科書,インプレス R&D(2008)
[8]. http://www.icom.co.jp/
[9]. 三菱電線工業時報　第 105 号(2008)

付録

ソースコード１のプログラムは提案手法１、ソースコード２のプログラムは提案手法２を実装したものである。

ソースコード１

```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>//常に変化する値、システム時刻
入

#define SIZE 1024

int avgGetRandom(int min,int max);//乱数範囲指定用関数
int carGetRandom(int min,int max);//乱数範囲指定用関数
int spdGetRandom(int min,int max);//乱数範囲指定用関数

int main(void){

int jisoku[SIZE];
double funsoku[SIZE];
int fun; //何分か入れる（単位：分）
int run[SIZE]; //車が走った距離
int daisu; //車の台数
int i,j,k=1,m;//ループカウント変数
int speed[SIZE];
int si[SIZE],sg[SIZE]; //四捨五入に使う変数
int spd[SIZE];

int rjisoku[SIZE];
double rfunsoku[SIZE];
int rfun; //何分か入れる（単位：分）
int rrun[SIZE]; //車が走った距離
int rdaisu; //車の台数
int ri,rj,rk=1,rm;//ループカウント変数
int dai;//代入用変数
int rsi[SIZE],rsg[SIZE]; //四捨五入に使う変数
int rspd[SIZE];

FILE *file;
file = fopen("test2.txt","w");
   //test.txt ファイルに書き込む準備.ファイルを開く f

      srand((unsigned)time( NULL ));//乱数

daisu = carGetRandom(1000,2000);
 //シミュレータ上の車の台数を決める
本番は min=2000 max=5000
for(i=1; i<=daisu; i++){ //車の台数分ループする
jisoku[i] = avgGetRandom(30,160);
    //各車 1 台 1 台の時速を出す ()内で最高、最低速度を指定
funsoku[i] = (double)(jisoku[i] / 60.0);
      //各車 1 台 1 台の分あたりの速度を出す
}

fun=100;
for(i=1 ; i<=daisu; i++){ //車の台数分ループする
for(j=10 ; j<=fun; j=j+10){
   //10 分ごとの進んだ距離を表示
speed[i] = funsoku[i];
run[k] = j * funsoku[i];      //入力した時間（単位：分）×　分速＝車が走った距離
if(run[k]<=200){
if(run[k] - 10 <= run[k] <= run[k] + 10) { /////渋滞情報の入出力

si[k] = run[k]; //車の現在位置 runk を si に代入
sg[k] = si[k] % 10;   //現在位置を 10 で割った余りを sg に代入

if(sg[k]>5){
si[k] = si[k] + (10-sg[k]);

for(m=si[k];m<=200;m=m+10){
spd[i] = spdGetRandom(50,160);
if(spd[i]<=20){
fprintf(file,"==混雑レベル[大]==¥n¥n");
}
else if(spd[i]<=40){
fprintf(file,"==混雑レベル[中]==¥n¥n");
}
else if(spd[i]<=60){
fprintf(file,"==混雑レベル[小]==¥n¥n");
}
else if(spd[i]<80){
fprintf(file,"==混雑レベル[渋滞なし]==¥n¥n");
}
else      if(spd[i]>=80){
fprintf(file,"==混雑レベル[混雑なし]==¥n¥n");
}
}

}else{
si[k] = si[k] - sg[k];
for(m=si[k];m<=200;m=m+10){
spd[i] = spdGetRandom(50,160);
fprintf(file,"%dkm 地点の情報を出力　",m);
if(spd[i]<=20){
fprintf(file,"==混雑レベル[大]==¥n¥n");
}
else if(spd[i]<=40){
fprintf(file,"==混雑レベル[中]==¥n¥n");
}
else if(spd[i]<=60){
fprintf(file,"==混雑レベル[小]==¥n¥n");
}
else if(spd[i]<80){
fprintf(file,"==混雑レベル[渋滞なし]==¥n¥n");
}
else      if(spd[i]>=80){
fprintf(file,"==混雑レベル[混雑なし]==¥n¥n");
}
}
}

}
k++;
}
else{
fp
```

```
k++;
}

}
fprintf(file,"¥n");
}

rdaisu = carGetRandom(2000,3000);
 //シミュレータ上の車の台数を決める
 本番は min=2000 max=5000
for(ri=1; ri<=rdaisu; ri++){ //車の台数分ループする
rjisoku[ri] = avgGetRandom(30,160);
   //各車 1 台 1 台の時速を出す ()内で最高、最低速度を指定
rfunsoku[ri] = (double)(rjisoku[ri] / 60.0);
//各車 1 台 1 台の分あたりの速度を出す
}

rfun=100;

for(ri=1 ; ri<=rdaisu; ri++){ //車の台数分ループする
for(rj=10 ; rj<=rfun; rj = rj+10){
   //10 分ごとの進んだ距離を表示
rrun[rk] = rj * rfunsoku[ri];
 //入力した時間（単位：分）×　分速＝車が走った距離
dai = rrun[rk];   //走った距離を代入する
dai = 200 - dai; //下りなので総延長 200 から引く
if(dai >= 0){
fprintf(file,"下り%d 台目の%d 分後の車の位置は、
%dkm 地点です¥n",ri,rj,dai);
rk++;
if(rrun[rk] - 10 <= rrun[rk] <= rrun[rk] + 10) {
rsi[rk] = dai;
rsg[rk] = rsi[rk] % 10;
if(rsg[rk] > 5){
rsi[rk] = rsi[rk] + (10 - rsg[rk]);
rspd[i] = spdGetRandom(50,160);
fprintf(file,"%dkm 地点の情報を出力　",m);

if(rspd[i]<=20){
fprintf(file,"==混雑レベル[大]==¥n¥n");
}
else if(rspd[i]<=40){
fprintf(file,"==混雑レベル[中]==¥n¥n");
}
else if(rspd[i]<=60){
fprintf(file,"==混雑レベル[小]==¥n¥n");
}
else if(rspd[i]<80){
fprintf(file,"==混雑レベル[渋滞なし]==¥n¥n");
}
else      if(rspd[i]>=80){
fprintf(file,"==混雑レベル[混雑なし]==¥n¥n");
}
}

}else{
rsi[rk] = rsi[rk] - rsg[rk];
for(m=rsi[rk];m>=0;m=m-10){
rspd[i] = spdGetRandom(50,160);
fprintf(file,"%dkm 地点の情報を出力　",m);

if(rspd[i]<=20){
fprintf(file,"==混雑レベル[大]==¥n¥n");
}
else if(rspd[i]<=40){
fprintf(file,"==混雑レベル[中]==¥n¥n");
}
else if(rspd[i]<=60){
fprintf(file,"==混雑レベル[小]==¥n¥n");
}
else if(rspd[i]<80){
fprintf(file,"==混雑レベル[渋滞なし]==¥n¥n");
}
else      if(rspd[i]>=80){
fprintf(file,"==混雑レベル[混雑なし]==¥n¥n");
}
}
}
}
}
}
else{
fprintf(file,"下り%d 台目の%d 分後の車の位置は、%dkm 地
点です¥n",ri,rj,0);
rk++;
}
}
fprintf(file,"¥n");
}

fclose(file);//ファイルを閉じる
return 0;
}



//乱数範囲指定用関数
int avgGetRandom(int min,int max)
{
return                min                +
(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}

int carGetRandom(int min,int max)
{
return                min                +
(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}

int spdGetRandom(int min,int max)
{
return                min                +
(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}
```

ソースコード２

```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>//常に変化する値、システム時刻

#define SIZE 10240
int  avgGetRandom(int  min,int  max);//乱数範囲指定用
```

```
関数
int carGetRandom(int min,int max);//乱数範囲指定用関数

int main(void){
int jisoku[SIZE];
double funsoku[SIZE];
int fun; //何分か入れる（単位：分）
int run[SIZE]; //車が走った距離
int daisu; //車の台数
int i,j,k=1,m;//ループカウント変数
int speed[SIZE];
int si[SIZE],sg[SIZE]; //四捨五入に使う変数
int info[SIZE];//距離地点における車の速度
int count[SIZE];//距離地点における車の台数
int kekka[SIZE];//平均速度
int kiro;

int rjisoku[SIZE];
double rfunsoku[SIZE];
int rfun; //何分か入れる（単位：分）
int rrun[SIZE]; //車が走った距離
int rdaisu; //車の台数
int ri,rj,rk=1,rm;//ループカウント変数
int dai;//代入用変数
int rsi[SIZE],rsg[SIZE]; //四捨五入に使う変数
int rinfo[SIZE];//距離地点における車の速度
int rcount[SIZE];//距離地点における車の台数
int rkekka[SIZE];//平均速度
int rkiro;

FILE *file;
file = fopen("test1.txt","w");  //test.txt ファイルに書き込む準備.ファイルを開く f
srand((unsigned)time( NULL ));//乱数

daisu = carGetRandom(2000,3000); //シミュレータ上の車の台数を決める 本番は min=2000 max=5000
for(i=1; i<=daisu; i++){ //車の台数分ループする
jisoku[i] = avgGetRandom(0,120);                //各車 1 台 1 台の時速を出す ()内で最高、最低速度を指定
funsoku[i] = (double)(jisoku[i] / 60.0);          // 各 車 1 台 1 台の分あたりの速度を出す
}


fun=100;

for(i=1 ; i<=daisu; i++){ //車の台数分ループする
for(j=10 ; j<=fun; j=j+10){                //10 分ごとの進んだ距離を表示
speed[i] = funsoku[i];
run[k] = j * funsoku[i];     //入力した時間（単位：分）×　分速＝車が走った距離
if(run[k]<=200){
if(run[k] - 10 <= run[k] <= run[k] + 10) { /////渋滞情報の入出力
si[k] = run[k]; //車の現在位置 runk を si に代入
sg[k] = si[k] % 10;  //現在位置を 10 で割った余りを sg に代入
if(sg[k]>5){
si[k] = si[k] + (10-sg[k]);

if(si[k] == 10){//もし、距離地点 10km なら
info[1] = info[1] + jisoku[i];
count[1]++;
}
else if(si[k] == 20){//もし、距離地点 20km なら
info[2] = info[2] + jisoku[i];
count[2]++;
}
else if(si[k] == 30){//もし、距離地点 30km なら
info[3] = info[3] + jisoku[i];
count[3]++;
}
else if(si[k] == 40){//もし、距離地点 20km なら
info[4] = info[4] + jisoku[i];
count[4]++;
}
else if(si[k] == 50){//もし、距離地点 30km なら
info[5] = info[5] + jisoku[i];
count[5]++;
}
else if(si[k] == 60){//もし、距離地点 20km なら
info[6] = info[6] + jisoku[i];
count[6]++;
}
else if(si[k] == 70){//もし、距離地点 30km なら
info[7] = info[7] + jisoku[i];
count[7]++;
}
else if(si[k] == 80){//もし、距離地点 20km なら
info[8] = info[8] + jisoku[i];
count[8]++;
}
else if(si[k] == 90){//もし、距離地点 30km なら
info[9] = info[9] + jisoku[i];
count[9]++;
}
else if(si[k] == 100){//もし、距離地点 20km なら
info[10] = info[10] + jisoku[i];
count[10]++;
}
else if(si[k] == 110){//もし、距離地点 30km なら
info[11] = info[11] + jisoku[i];
count[11]++;
}
else if(si[k] == 120){//もし、距離地点 20km なら
info[12] = info[12] + jisoku[i];
count[12]++;
}
else if(si[k] == 130){//もし、距離地点 30km なら
info[13] = info[13] + jisoku[i];
count[13]++;
}
else if(si[k] == 140){//もし、距離地点 20km なら
info[14] = info[14] + jisoku[i];
count[14]++;
}
else if(si[k] == 150){//もし、距離地点 30km なら
info[15] = info[15] + jisoku[i];
count[15]++;
}
else if(si[k] == 160){//もし、距離地点 20km なら
info[16] = info[16] + jisoku[i];
count[16]++;
}
else if(si[k] == 170){//もし、距離地点 30km なら
info[17] = info[17] + jisoku[i];
count[17]++;
```

```
}
else if(si[k] == 180){//もし、距離地点 30km なら
info[18] = info[18] + jisoku[i];
count[18]++;
else if(si[k] == 190){//もし、距離地点 20km なら
info[19] = info[19] + jisoku[i];
count[19]++;
}
else if(si[k] == 200){//もし、距離地点 30km なら
info[20] = info[20] + jisoku[i];
count[20]++;
}
}
else{
si[k] = si[k] - sg[k];
if(si[k] == 10){//もし、距離地点 10km なら
info[1] = info[1] + jisoku[i];
count[1]++;
}
else if(si[k] == 20){//もし、距離地点 20km なら
info[2] = info[2] + jisoku[i];
count[2]++;
}
else if(si[k] == 30){//もし、距離地点 30km なら
info[3] = info[3] + jisoku[i];
count[3]++;
}
else if(si[k] == 40){//もし、距離地点 20km なら
info[4] = info[4] + jisoku[i];
count[4]++;
}
else if(si[k] == 50){//もし、距離地点 30km なら
info[5] = info[5] + jisoku[i];
count[5]++;
}
else if(si[k] == 60){//もし、距離地点 20km なら
info[6] = info[6] + jisoku[i];
count[6]++;
}
else if(si[k] == 70){//もし、距離地点 30km なら
info[7] = info[7] + jisoku[i];
count[7]++;
}
else if(si[k] == 80){//もし、距離地点 20km なら
info[8] = info[8] + jisoku[i];
count[8]++;
}
else if(si[k] == 90){//もし、距離地点 30km なら
info[9] = info[9] + jisoku[i];
count[9]++;
}
else if(si[k] == 100){//もし、距離地点 20km なら
info[10] = info[10] + jisoku[i];
count[10]++;
}
else if(si[k] == 110){//もし、距離地点 30km なら
info[11] = info[11] + jisoku[i];
count[11]++;
}
else if(si[k] == 120){//もし、距離地点 20km なら
info[12] = info[12] + jisoku[i];
count[12]++;
}
else if(si[k] == 130){//もし、距離地点 30km なら
info[13] = info[13] + jisoku[i];
count[13]++;
}
else if(si[k] == 140){//もし、距離地点 20km なら
info[14] = info[14] + jisoku[i];
count[14]++;
}
else if(si[k] == 150){//もし、距離地点 30km なら
info[15] = info[15] + jisoku[i];
count[15]++;
}
else if(si[k] == 160){//もし、距離地点 20km なら
info[16] = info[16] + jisoku[i];
count[16]++;
}
else if(si[k] == 170){//もし、距離地点 30km なら
info[17] = info[17] + jisoku[i];
count[17]++;
}
else if(si[k] == 180){//もし、距離地点 30km なら
info[18] = info[18] + jisoku[i];
count[18]++;
}
else if(si[k] == 190){//もし、距離地点 20km なら
info[19] = info[19] + jisoku[i];
count[19]++;
}
else if(si[k] == 200){//もし、距離地点 30km なら
info[20] = info[20] + jisoku[i];
count[20]++;
}
}

}
k++;
}
else{
k++;
}

}
}
printf("============= 上 り 車 線 渋 滞 情 報 処 理
==============¥n");
for(i=1; i<=20; i++){
kekka[i] = info[i] / count[i]; //距離地点別における平均速
度を求める
}
kiro=10;
for(i=1; i<=20; i++){
printf("%dkm 地点の渋滞情報を出力　",kiro,kekka[i]);
if(kekka[i]<=20){
printf("==混雑レベル[大]==¥n");
}
else　　if(kekka[i]<=40){
printf("==混雑レベル[中]==¥n");
}
else　　if(kekka[i]<=60){
printf("==混雑レベル[小]==¥n");
}
else　　if(kekka[i]<80){
printf("==混雑レベル[渋滞なし]==¥n");
}
else　　if(kekka[i]>=80){
printf("==混雑レベル[混雑なし]==¥n");
}
```

```
kiro=kiro+10;//10km ごとを出すため足している
}
printf("==========================================¥n");



rdaisu = carGetRandom(8000,9000); //シミュレータ上の車の台数を決める
for(ri=1; ri<=rdaisu; ri++){ //車の台数分ループする
rjisoku[ri] = avgGetRandom(0,120);                    //各車 1 台 1 台の時速を出す ()内で最高、最低速度を指定
rfunsoku[ri] = (double)(rjisoku[ri] / 60.0);      // 各 車 1 台 1 台の分あたりの速度を出す
}

rfun=100;
for(ri=1 ; ri<=rdaisu; ri++){ //車の台数分ループする
for(rj=10 ; rj<=rfun; rj = rj+10){          //10 分ごとの進んだ距離を表示
rrun[rk] = rj * rfunsoku[ri];      //入力した時間（単位：分）×　分速＝車が走った距離
dai = rrun[rk];  //走った距離を代入する
dai = 200 - dai; //下りなので総延長 200 から引く
if(dai >= 0){
rk++;
if(rrun[rk] - 10 <= rrun[rk] <= rrun[rk] + 10) {
rsi[rk] = dai;
rsg[rk] = rsi[rk] % 10;
if(rsg[rk] > 5){
rsi[rk] = rsi[rk] + (10 - rsg[rk]);

if(rsi[rk] == 10){//もし、距離地点 10km なら
rinfo[1] = rinfo[1] + rjisoku[ri];
rcount[1]++;
}
else if(rsi[rk] == 20){//もし、距離地点 20km なら
rinfo[2] = rinfo[2] + rjisoku[ri];
rcount[2]++;
}
else if(rsi[rk] == 30){//もし、距離地点 30km なら
rinfo[3] = rinfo[3] + rjisoku[ri];
rcount[3]++;
}
else if(rsi[rk] == 40){//もし、距離地点 20km なら
rinfo[4] = rinfo[4] + rjisoku[ri];
rcount[4]++;
}
else if(rsi[rk] == 50){//もし、距離地点 30km なら
rinfo[5] = rinfo[5] + rjisoku[ri];
rcount[5]++;
}
else if(rsi[rk] == 60){//もし、距離地点 20km なら
rinfo[6] = rinfo[6] + rjisoku[ri];
rcount[6]++;
}
else if(rsi[rk] == 70){//もし、距離地点 30km なら
rinfo[7] = rinfo[7] + rjisoku[ri];
rcount[7]++;
}
else if(rsi[rk] == 80){//もし、距離地点 20km なら
rinfo[8] = rinfo[8] + rjisoku[ri];
rcount[8]++;
}
else if(rsi[rk] == 90){//もし、距離地点 30km なら
rinfo[9] = rinfo[9] + rjisoku[ri];
rcount[9]++;
}
else if(rsi[rk] == 100){//もし、距離地点 20km なら
rinfo[10] = rinfo[10] + rjisoku[ri];
rcount[10]++;
}
else if(rsi[rk] == 110){//もし、距離地点 30km なら
rinfo[11] = rinfo[11] + rjisoku[ri];
rcount[11]++;
}
else if(rsi[rk] == 120){//もし、距離地点 20km なら
rinfo[12] = rinfo[12] + rjisoku[ri];
rcount[12]++;
}
else if(rsi[rk] == 130){//もし、距離地点 30km なら
rinfo[13] = rinfo[13] + rjisoku[ri];
rcount[13]++;
}
else if(rsi[rk] == 140){//もし、距離地点 20km なら
rinfo[14] = rinfo[14] + rjisoku[ri];
rcount[14]++;
}
else if(rsi[rk] == 150){//もし、距離地点 30km なら
rinfo[15] = rinfo[15] + rjisoku[ri];
rcount[15]++;
}
else if(rsi[rk] == 160){//もし、距離地点 20km なら
rinfo[16] = rinfo[16] + rjisoku[ri];
rcount[16]++;
}
else if(rsi[rk] == 170){//もし、距離地点 30km なら
rinfo[17] = rinfo[17] + rjisoku[ri];
rcount[17]++;
}
else if(rsi[rk] == 180){//もし、距離地点 30km なら
rinfo[18] = rinfo[18] + rjisoku[ri];
rcount[18]++;
}
else if(rsi[rk] == 190){//もし、距離地点 20km なら
rinfo[19] = rinfo[19] + rjisoku[ri];
rcount[19]++;
}
else if(rsi[rk] == 200){//もし、距離地点 30km なら
rinfo[20] = rinfo[20] + rjisoku[ri];
rcount[20]++;
}

}else{
rsi[rk] = rsi[rk] - rsg[rk];
if(rsi[rk] == 10){//もし、距離地点 10km なら
rinfo[1] = rinfo[1] + rjisoku[ri];
rcount[1]++;
}
else if(rsi[rk] == 20){//もし、距離地点 20km なら
rinfo[2] = rinfo[2] + rjisoku[ri];
rcount[2]++;
}
else if(rsi[rk] == 30){//もし、距離地点 30km なら
rinfo[3] = rinfo[3] + rjisoku[ri];
rcount[3]++;
}
else if(rsi[rk] == 40){//もし、距離地点 20km なら
rinfo[4] = rinfo[4] + rjisoku[ri];
rcount[4]++;
```

```
}
else if(rsi[rk] == 50){//もし、距離地点 30km なら
rinfo[5] = rinfo[5] + rjisoku[ri];
rcount[5]++;
}
else if(rsi[rk] == 60){//もし、距離地点 20km なら
rinfo[6] = rinfo[6] + rjisoku[ri];
rcount[6]++;
}
else if(rsi[rk] == 70){//もし、距離地点 30km なら
rinfo[7] = rinfo[7] + rjisoku[ri];
rcount[7]++;
}
else if(rsi[rk] == 80){//もし、距離地点 20km なら
rinfo[8] = rinfo[8] + rjisoku[ri];
rcount[8]++;
}
else if(rsi[rk] == 90){//もし、距離地点 30km なら
rinfo[9] = rinfo[9] + rjisoku[ri];
rcount[9]++;
}
else if(rsi[rk] == 100){//もし、距離地点 20km なら
rinfo[10] = rinfo[10] + rjisoku[ri];
rcount[10]++;
}
else if(rsi[rk] == 110){//もし、距離地点 30km なら
rinfo[11] = rinfo[11] + rjisoku[ri];
rcount[11]++;
}
else if(rsi[rk] == 120){//もし、距離地点 20km なら
rinfo[12] = rinfo[12] + rjisoku[ri];
rcount[12]++;
}
else if(rsi[rk] == 130){//もし、距離地点 30km なら
rinfo[13] = rinfo[13] + rjisoku[ri];
rcount[13]++;
}
else if(rsi[rk] == 140){//もし、距離地点 20km なら
rinfo[14] = rinfo[14] + rjisoku[ri];
rcount[14]++;
}
else if(rsi[rk] == 150){//もし、距離地点 30km なら
rinfo[15] = rinfo[15] + rjisoku[ri];
rcount[15]++;
}
else if(rsi[rk] == 160){//もし、距離地点 20km なら
rinfo[16] = rinfo[16] + rjisoku[ri];
rcount[16]++;
}
else if(rsi[rk] == 170){//もし、距離地点 30km なら
rinfo[17] = rinfo[17] + rjisoku[ri];
rcount[17]++;
}
else if(rsi[rk] == 180){//もし、距離地点 30km なら
rinfo[18] = rinfo[18] + rjisoku[ri];
rcount[18]++;
}
else if(rsi[rk] == 190){//もし、距離地点 20km なら
rinfo[19] = rinfo[19] + rjisoku[ri];
rcount[19]++;
}
else if(rsi[rk] == 200){//もし、距離地点 30km なら
rinfo[20] = rinfo[20] + rjisoku[ri];
rcount[20]++;
}
}
}
else{
rk++;
}
}
}
printf("¥n¥n==========下り車線渋滞情報処理
==========¥n");
rkekka[1]=rinfo[1] / rcount[1];
rkekka[2]=rinfo[2] / rcount[2];
rkekka[3]=rinfo[3] / rcount[3];
rkekka[4]=rinfo[4] / rcount[4];
rkekka[5]=rinfo[5] / rcount[5];
rkekka[6]=rinfo[6] / rcount[6];
rkekka[7]=rinfo[7] / rcount[7];
rkekka[8]=rinfo[8] / rcount[8];
rkekka[9]=rinfo[9] / rcount[9];
rkekka[10]=rinfo[10] / rcount[10];
rkekka[11]=rinfo[11] / rcount[11];
rkekka[12]=rinfo[12] / rcount[12];
rkekka[13]=rinfo[13] / rcount[13];
rkekka[14]=rinfo[14] / rcount[14];
rkekka[15]=rinfo[15] / rcount[15];
rkekka[16]=rinfo[16] / rcount[16];
rkekka[17]=rinfo[17] / rcount[17];
rkekka[18]=rinfo[18] / rcount[18];
rkekka[19]=rinfo[19] / rcount[19];
rkekka[20]=rinfo[20] / rcount[20];
rkiro=10;
for(i=1; i<=20; i++){
printf("%dkm 地点の渋滞情報を出力
",rkiro,rkekka[i]);
if(rkekka[i]<=20){
printf("==混雑レベル[大]==¥n");
}
else if(rkekka[i]<=40){
printf("==混雑レベル[中]==¥n");
}
else if(rkekka[i]<=60){
printf("==混雑レベル[小]==¥n");
}
else if(rkekka[i]<80){
printf("==混雑レベル[渋滞なし]==¥n");
}
else if(rkekka[i]>=80){
printf("==混雑レベル[混雑なし]==¥n");
}
rkiro=rkiro+10;//10km ごとを出すため足している
}
printf("==================================
============¥n");
fclose(file);//ファイルを閉じる
return 0;
}
//乱数範囲指定用関数
int avgGetRandom(int min,int max)
{
return min +
(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}
int carGetRandom(int min,int max)
{
return min +
```

```
(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}
```