修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　　情報理工学 研究科　　情報・通信工学 専攻　博士前期課程		
氏　　名	吉川 潤	学籍番号	1331113
論 文 題 目	狭域交通情報共有のための車車間通信における車両位置情報に基づく 効率的な中継転送方式の提案		

要　旨

近年，日本では人々の生活において重要な交通手段である自動車とその交通事情は高水準な道路建設と安全対策の進展によって快適性と安全性が増してきている．しかし，平成 24 年のデータによると交通事故の発生件数 66 万件，負傷者数 82 万人，死者 4000 人と他の交通機関である鉄道，飛行機，船舶と比較すると依然として安全とは呼べない水準である．問題はそれだけではなく，通勤時間帯における交通渋滞も深刻で，大規模な渋滞が慢性的に発生し経済的な損害と環境に悪影響を及ぼしている．この様な現状を受けて，社会への貢献のテーマとして自動車による移動の，快適性・安全性・エコロジーを挙げる．そこで車車間通信を用いて，渋滞情報や急ブレーキ情報，緊急車両情報などのあらゆる情報を車両間で共有することで，自動車の快適性と安全性の向上，環境汚染の低減を実現したい．

そのため筆者は，車車間通信により渋滞や事故等の狭域な交通情報を周辺車両で効率的に共有させるためのパケット中継転送方式の検討を行った．従来からのパケット中継転送方式では，中継時にほぼ同じ地理的な位置に存在する複数車両の通信が衝突するという問題点があり，筆者らはこの問題を解決するため，実用化が進みつつある 700MHz 帯の電波を使った安全運転支援システムにより車両間の位置情報が周辺車両で共有されることを想定し，それに基づいてほぼ同じ位置に存在する中継車両候補の送信順序を明確に区別し中継の衝突を抑制する方式を考案した．加えて，提案方式では交差点車両に対する特別な優先度の割当てや，通信混雑時には中継車両台数を限定する機能を検討した．

提案方式に対して都市部の交通を模したシミュレーションを行った結果，既存方式よりも最大で拡散率が 35%向上し，通信回数を 20.5%削減，加えて，遅延時間も最大で 55.3%削減することができた．

最後に，提案方式の評価で得た知見から，今後の課題や展望を述べる．

2014 年度

狭域交通情報共有のための車車間通信における車両位置情報に基づく効率的な中継転送方式の提案

学籍番号　1331113

氏名　吉川 潤

指導教員 小花 貞夫

副指導教員 沼尾 雅之

電気通信大学院　情報理工学研究科

2015 年 3 月 6 日

狭域交通情報共有のための車車間通信における車両位置情報に基づく効率的な中継転送方式の提案

吉川 潤

概要

近年，日本では人々の生活において重要な交通手段である自動車とその交通事情は高水準な道路建設と安全対策の進展によって快適性と安全性が増してきている.しかし，平成 24 年のデータによると交通事故の発生件数 66 万件，負傷者数 82 万人，死者 4,000 人と他の交通機関である鉄道，飛行機，船舶と比較すると依然として安全とは呼べない水準である.問題はそれだけではなく，通勤時間帯における交通渋滞も深刻で，大規模な渋滞が慢性的に発生し経済的な損害と環境に悪影響を及ぼしている.この様な現状を受けて，社会への貢献のテーマとして自動車による移動の，快適性・安全性・エコロジーを挙げる.そこで車車間通信を用いて，渋滞情報や急ブレーキ情報，緊急車両情報などのあらゆる情報を車両間で共有することで，自動車の快適性と安全性の向上，環境汚染の低減を実現したい.

そのため，車車間通信により渋滞や事故等の狭域な交通情報を周辺車両で効率的に共有するためのパケット中継転送方式の検討を行った.従来からのパケット中継転送方式では，中継時にほぼ同じ地理的な位置に存在する複数車両の通信が衝突するという問題点があり，本論文ではこの問題を解決するため，実用化が進みつつある 700MHz 帯の電波を使った安全運転支援システムにより車両間の位置情報が周辺車両で共有されることを想定し，それに基づいてほぼ同じ位置に存在する中継車両候補の送信順序を明確に区別し中継の衝突を抑制する方式を考案した.提案方式では交差点車両に対する特別な優先度の割当てや，通信混雑時には中継車両台数を限定する機能も検討した.提案方式に対して都市部の交通を模したシミュレーションにより，既存方式よりも最大で拡散率が 35%向上し，通信回数を 20.5%削減，遅延時間も最大で 55.3%削減することができ，提案方式の有効性を確認した.

目次

図目次

表目次

1. 序論

現代の社会において，自動車は日々の生活とは切っても切れないものとなっている.自宅からあらゆる目的地まで移動することのできる自動車は，柔軟性の高い乗り物として都市部や地方部でなくてはならないものであり，産業においても自動車の輸送は，物流の要として日本経済を支えている.一方で，自動車を利用する上ではまだまだ不便な点が多く存在する.例えば，ドライバ同士が車線変更や右左折，停止などをしたい時にそれを伝える手段としてウィンカのみしか存在せず，ドライバが要求する行動が車線変更なのか右左折なのか判断が付きにくいといったような問題が起こりうる.そのためドライバ同士の望む行動を共有することでより快適な運転をサポートできる.それだけではなく渋滞情報や急ブレーキ情報，緊急車両情報など自動車を運転している時に知っておくべきあらゆる情報を共有することでより安全に快適でエコロジーな車社会を実現できる.そのような社会の実現のために高度道路交通システムでは，車車間通信により，お互いの位置・速度情報等を頻繁に交換して衝突を防止するシステムの研究開発[1]が行われてきているが，今後は，安全運転のみならず，エコドライブや運転の快適性利便性向上を図るシステムの実現も強く期待される.本論文では，ドライバが道路や交通の状況に応じて走行できるように，車車間通信により渋滞や事故等の局地的な交通情報を周辺車両で効率的に共有するために，位置情報を利用した順序付けによる中継車両選択方式を新たに提案し，実際の道路状況を模したシミュレーションにより提案方式の有効性を検証する.

以下，第二章では，研究の背景を述べ，三章では交通情報共有を目的とした先行研究と既存の中継車両選択方式について述べる.次いで, 第四章では, 提案方式の目的と機能詳細, 第五章ではシミュレーション評価の詳細，第六章ではシミュレーション評価の結果と考察, 第七章では，結論と今後の課題について述べる.

2. 研究の背景

2.1. 高度道路交通システム

高度道路交通システム(ITS:Intelligent Transport Systems）とは，人や道路，自動車の間で情報の受発信を行い，道路交通が抱える事故や渋滞，環境対策など，様々な課題を解決し，常に最先端の情報通信や制御技術を活用して，道路交通の最適化を図ることを目的としており，国土技術政策総合研究所に定められる ITS の研究分野には以下のものがある.

- ナビゲーションシステムの高度化(VICS 等によるナビゲーションシステムの高度化)
- 自動料金収受システム(料金所等でのノンストップ化)
- 安全運転の支援(AHS 等による危険警告・自動運転)
- 交通管理の最適化(経路誘導，信号制御)
- 道路管理の効率化(特殊車両管理，通行規制状況の提供)
- 公共交通の支援(公共交通の運行状況の提供)
- 商用車の効率化(商用車の運行管理支援，連続自動運転)
- 歩行者等への経路，施設案内
- 緊急車両の運行支援(緊急時自動通報，災害・事故発生時の状況などの伝達等)

中でも，カーナビゲーション，VICS(Vehicle Information and Communication System)，ETC(electronic toll collection)，ASV(Advanced Safety Vehicle）等，ITS 個別要素技術の研究開発が推進され，これらはカーナビゲーション市場の成長とともに日本の ITS の成功事例として世界に知られている.ほかにも，信号制御や道路防災などの道路交通管理分野，またバスロケーションシステムや PTPS(Public Transportation Priority System）等の公共交通分野，さらには携帯電話を使ったテレマティクスサービス分野等様々な分野で，着実な展開・実用化が進んだ.今後の取り組みの方向性として，「安全・安心」「環境・効率」「快適・利便」を基本概念とする「ITS 推進の指針」が，日本 ITS 推進会議によりまとめられた.

この指針が，2006 年 1 月の「IT 新改革戦略」に反映され，ITS は安全・環境・利便達成に貢献する技術として位置づけられ，「世界一安全な道路交通社会」を目指すインフラ協調安全運転支援の実用化プロジェクトが官民連携のもと進められている.

2.2. 車車間通信と路車間通信

自動車向けの無線通信の形態は，車車間通信と路車間通信，セルラ網を利用するものがある.車両同士が車載器を通じて直接情報をやりとりするものを車車間通信，車両が路側機と情報をやりとりするものを路車間通信と呼ぶ.

2.2.1. 車車間通信

車車間通信は，車両間での情報通信を行うため，各車両が車載器を搭載する必要がある.一方で，インフラ整備などの設備投資を必要とせずに実現可能である.通信のトラフィックは情報を交換する車両の数に応じて増加するため，車両数が増えた場合には効率的に情報交換が行われない可能性がある.また，通信を行う双方の車両が移動している場合，ドップラ効果や隠れ端末問題などが生じることが考えられる.加えて車載器を搭載する車両が満足に普及していない状態では，車車間通信による効率的な情報の共有が難しいという欠点も挙げられる.

現状では，安全運転支援のための車車間通信システムの実現に目途がついているが自動運転の実現に向けた交通情報の共有が強く期待さる.車車間通信を実現するための課題として，限られた周波数帯域を用いて多数の車両と情報を交換しなければならない点や，車車間通信で利用する情報に付加する GPS の座標の精度向上，使用すべきアンテナの方式の検討などが挙げられる

2.2.2. 路車間通信

路車間通信は，車両と路側機間での情報通信を行うため，車載器の搭載とインフラ設備の整備が必要であり，路側機が設置されていないエリアではサービスを受けることができない.しかし車車間通信とは異なり，移動しない路側機を利用するため，大容量なデータの転送や車両混雑時でも効率的に情報交換を行うことができる.更に，車車間通信とは異なり路車間通信は車載器の普及率が高くない状況でもある一定の効果が表れる点も特徴である.一般的な技術としては ETC や VICS などが挙げられ，現在 ITS として実用段階に入っているものは路車間通信を利用したものが多い.

2.2.3. セルラ網の利用

セルラ網で交通情報の共有を行う場合，既存の通信インフラを利用できるという利点が

ある.一方で，集められたデータは中央のサーバーで処理を行い，各車両に情報を伝達していく必要があるため，遅延が大きくなりがちであり，急制動などの緊急度の高い情報には対応できない問題点がある.また，セルラ網の通信トラフィックを圧迫するという面からも実現には課題が多いと言える.

2.3. 交通情報共有システムとその課題

交通情報共有システムとは，車車間通信や路車間通信を用いて車両が走行中に捕捉した様々な交通事象に関する情報を共有することで円滑で快適な運転を実現する仕組みである.共有する情報は事故・渋滞情報や緊急車両情報，路面情報，急ブレーキ情報，ドライバリクエストなどを想定し，情報の種類や位置データなどの小容量データを中心とする.また複数の車両が中継を行うことで，一回の電波の送信では届かない遠くの車両にも情報を伝達することができる.交通情報共有システムの概要を以下の図 1 に示す.

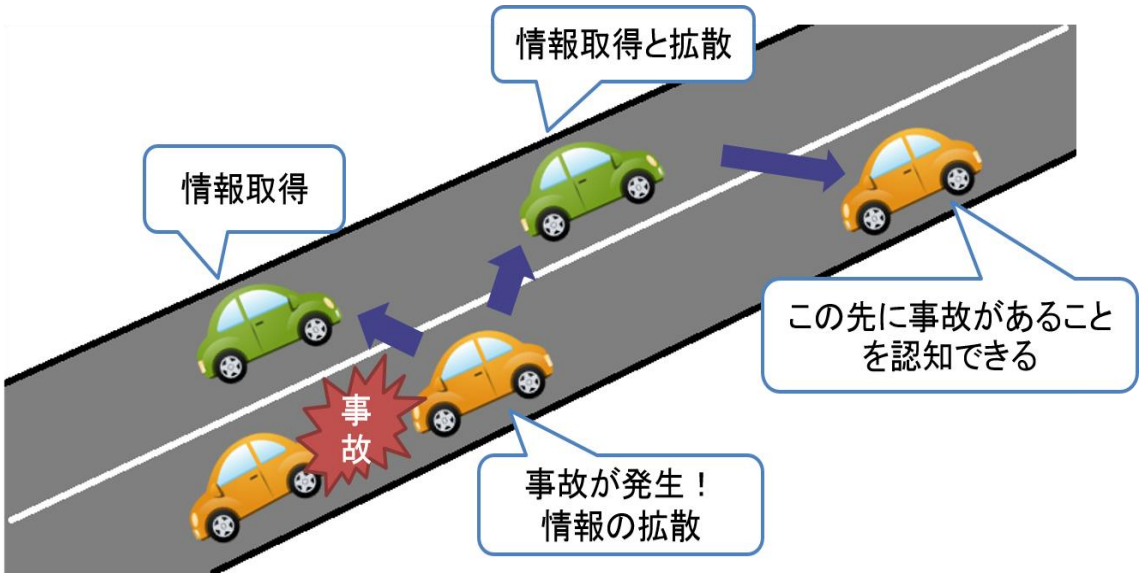

図 1 交通情報共有方式の概要

一般的な交通情報共有システムの主な課題としては以下の 3 点があげられる.

- 情報の効率的な拡散

配布する情報が多い場合には，すべての車両が受信した情報を他の車両に転送を行うと通信のトラフィックが圧迫され情報の拡散が適切に行われないことが想定される.そのため,各車両は情報を高速に拡散させると共に通信のトラフィックを抑制するために，情報を転送する車両を適切に選択する必要がある.

- 緊急度と配布範囲制御

配布させる情報の種類によって緊急度や配布範囲が異なるため，緊急度の高い情報の優先的な転送や，配布範囲内でのみ情報を拡散させる制御を行い，しかるべき情報をタイムリに適切な範囲に配布する必要がある.

- 多様なトラフィック環境への対応

道路を走行する自動車の密度は時間帯によって大きく異なる.特に，深夜・早朝や車載機の普及度が低い場合では電波の到達範囲内に他の車両が存在しない場合が考えられる.そのため，キャリアンドフォワード(各車両は受信した情報を一定期間保持し，移動しながら転送を繰り返す）等を行い，多様な車両の走行環境に対応する必要がある.

2.4. ITS への国際社会の取り組み

車車間等の移動体における通信では主に無線 LAN 通信規格である IEEE 802.11p が使用される.特に米国や欧州では，IEEE 802.11p に準拠した 5.9GHz 帯の周波数を ITS（高度交通システム）向けに採用し，車車間通信などに活用しようという動きがあり，製品の開発が進められている.一方で，日本国内では，5.8GHz 帯を使った ETC が普及している.また ETC の通信技術を利用し，ETC に加えて渋滞情報などが受け取れる DSRC 対応情報サービス ITS スポットも始まっている．加えて，電波を所管する総務省では，地上アナログ放送の終了で空いた 700MHz 帯の周波数帯域から 10MHz が ITS 向けに割り当て済みであり，総務省が実施した実験では，交差点から 180m 程度の通信距離が確保できることから，安全運転支援の車車間通信システムに使う方針を決めている.

2.4.1. 通信方式(IEEE 802.11p)と標準 MAC プロトコル

2.4.1.1. IEEE 802.11p の概要

802.11p は，IEEE802.11 規格に移動体通信環境(Wireless Access in Vehicular Environment)向けの拡張を施したものである.この拡張には高速で移動する車両と路側インフラとの協調も含まれており 5.9GHz(5.85-5.925Ghz)の ITS バンドを用いるものである.802.11p は DSRC(Dedicated Short-Range Communications)と呼ばれる狭域通信から CALM(Communications Access for Land Mobiles）と呼ばれる広域通信システムを対象としており ETC 等の決済システムや交通安全サービス，広告などの商取引での利用が想定される.

国際標準化機構（ISO:International Organization for Standardization）により定義された，通信機能の階層化モデルである OSI 参照モデルを用いる場合，IEEE 802.11p ではレイヤ 1 の物理層とレイヤ 2 のデータリンクレイヤの MAC レイヤについて標準化を行っている.MAC レイヤでは主にデータリンクレイヤ内の通信の分散制御や集中制御，ネットワーク媒体への信号の送出タイミングの決定について取り扱っている.OSI 参照モデルの概要を以下の表 1，また MAC レイヤで扱う MAC フレームの基本フォーマットを図 2 に示す.

表 1 OSI 参照モデル

OSI参照モデル	
アプリケーション層	具体的な通信サービス
プレゼンテーション層	データの表現方法
セッション層	通信プログラム間の通信の開始から終了までの手順
トランスポート層	ネットワークの端から端までの通信管理
ネットワーク層	ネットワークにおける通信経路の選択
データリンク層(MAC副層,LLC副層)	直接的（隣接的）に接続されている通信機器間の信号の受け渡し
物理層	物理的な接続の規定

引用:Wikipedia(OSI参照モデル)より

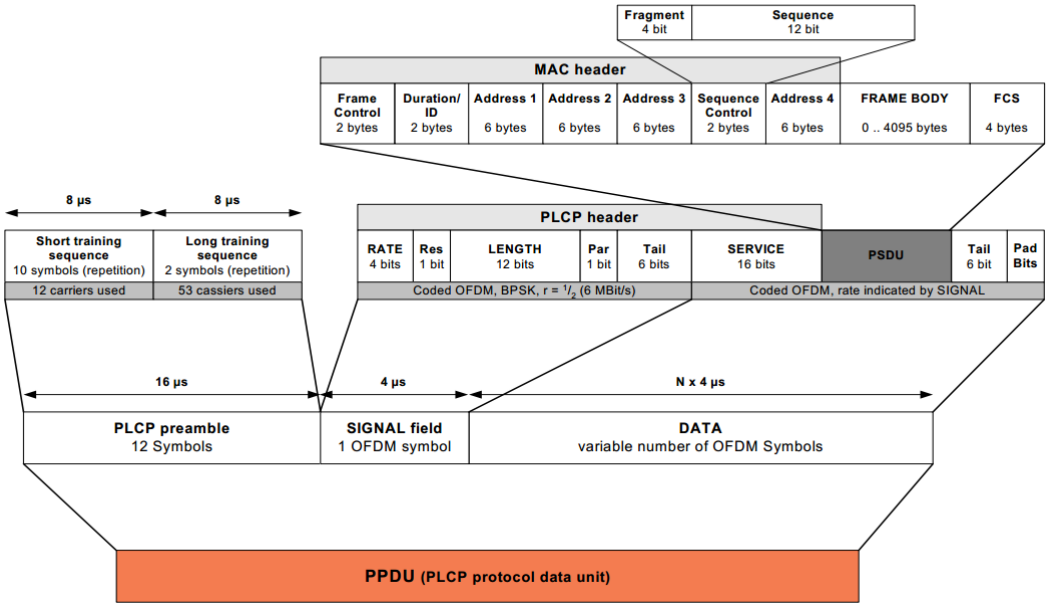

図 2　802.11MAC フレームフォーマット

引用:ローデ・シュワルツ・ジャパン(802.11p 測定の紹介)

(http://www.rohde-schwarz.co.jp/download/jp/an/1MA152_1J.pdf)

2.4.1.2. IEEE 802.11p の機能

IEEEE 802.11p は基本的に IEEE 802.11a の基本的な機能を受け継いでおり, CSMA/CA による制御方式や OFDM といった変調方式が用いられる.特に IEEE 802.11p では同じく 5Ghz 帯を使っている IEEE802.11a の物理層と同様の仕様となっている.

- OFDM

  OFDM(Orthogonal Frequency Division Multiplexing(直交波周波数分割多重))はデジタル変調方式の一種であり，日本国内の第 4 世代セルラ網の LTE(Long Term Evolution)，地上波デジタル放送や無線 LAN(IEEE 802.11a)などの伝送方式に採用されている.

- CSMA/CA

  CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance（搬送波感知多重アクセス／衝突回避方式）)は IEEE 802.11a/b/g および IEEE 802.11p におけるメディアアクセス制御(MAC)の手順として使われている.

  IEEE 802.11p における通信では限られた無線周波数帯を複数の端末で共有しなけれ

ばならない.そのため，各端末間での通信の衝突を防ぐための制御方式である.CSMA/CA の制御手順は以下の 3 つのプロセスによって実現される

① Carrier Sense(搬送波感知)
送信したいフレームがある端末は，送信を開始する前に自分以外の端末が既に通信を行っていないかを確認する.

② Multiple Access(多重アクセス)
Carrier Sense を行った結果，他の端末が通信を行っていなければ同一のチャンネルを共用し，通信を開始する.

③ Collision Avoidance(衝突回避)
Carrier Sense を行った結果，既に他の端末が通信を行っていた場合，その通信が終了するのを待つ.同様にある端末の通信終了を待っている端末が複数ある場合，すぐに送信するとフレームが衝突してしまう可能性があるので，端末ごとに異なる待機時間を設定し，その待機時間終了後に送信を開始する.待機時間終了前に他の端末が新たに通信を開始した場合は，同様に通信終了を待ち自端末が送信開始できるタイミングを計る．また，長時間にわたり自端末が情報を送信できない事態を防ぐために，待機時間は徐々に短縮されていく.

- 隠れ端末問題とさらし端末問題

隠れ端末問題とさらし端末問題とは IEEE 802.11 などの通信プロトコルで発生するフレーム衝突に関する課題である.それぞれの問題については以下の図 3 および図 4 を用いて説明する.

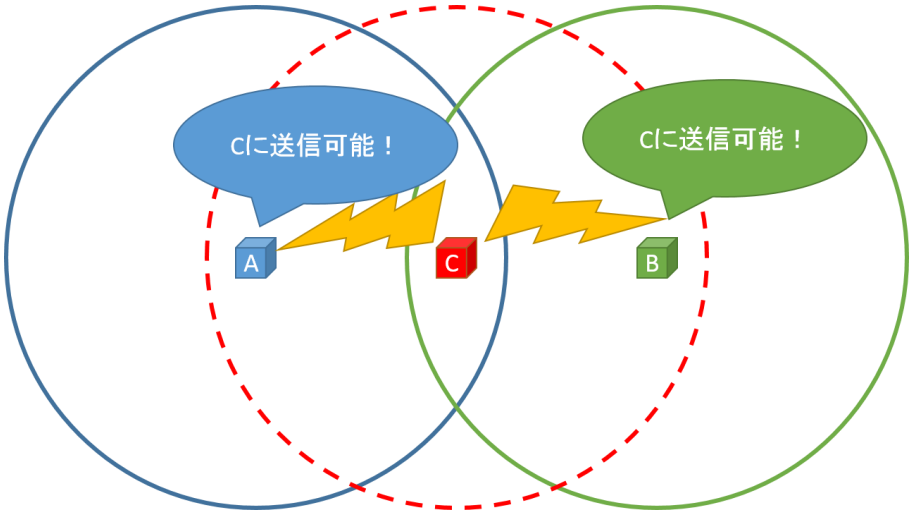

図 3　隠れ端末問題

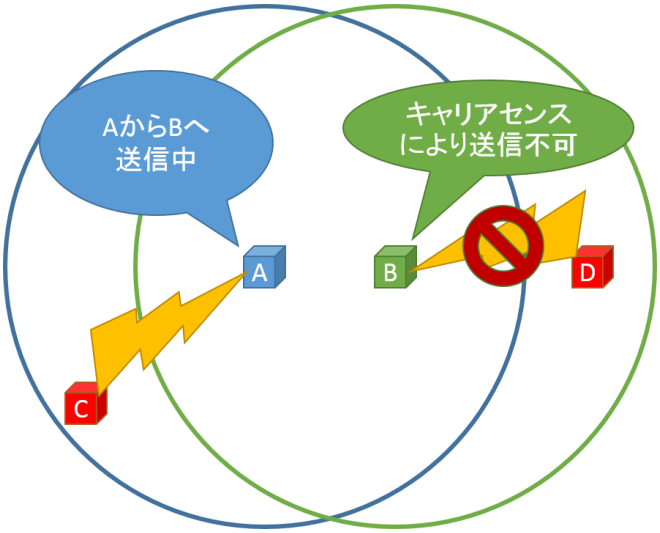

図 4　さらし端末問題

① 隠れ端末問題

隠れ端末とは，端末同士が互いの信号の到達範囲外にあることを意味し，隠れ端末問題とはAとBというお互いに通信範囲に存在しない端末がC端末にデータを転送しようとした場合にフレームが衝突する問題である.図 3 を用いて説明すると，前述の CSMA/CA における Carrier Sense プロセスにおいて端末 A および B は互いに相手の通信状況を把握することができない．そのため見かけ上端末 C への通信は可能だと考える．結果として，受信端末 C では同時に搬送波を検知してしまうためデータを受け取ることができない.隠れ端末状態の解決方法は RTS/CTS 方式により実現できるが，オーバヘッドが発生するためスループットが低下する可能性がある.

② さらし端末問題

さらし端末とはお互いの端末が通信可能範囲内にある場合に，それらの端末をさらし端末と表現する.さらし端末問題について図 4 を用いて説明すると，お互いにさらし端末である A と B において A が既にデータを送信中であることを B が Carrier Sense することで D にデータを送信することができずスループットが低下する可能性がある.さらし端末問題に対してはさらし端末状態にある端末同士では異なる周波数を用いてデータを送信することでスループットの低下を防ぐことができる.

3. 先行研究

3.1. 先行研究の概要

車車間通信を利用して交通情報の共有を目的とする研究は数多く存在するが，その通信方法はそれぞれ異な再ブロードキャスト[2-8,12-14]を用いた研究, マルチキャストもしくはユニキャストを用いた研究[9-11]に大別されるが，車車間通信において車両の急制動，安全運転支援や渋滞情報などの運転快適性を向上させるための通信では，通信の対象は特定の車両ではなく，情報を拡散させる範囲内に存在する不特定多数の車両へ伝達する必要がある.また各車両が高速に移動していることを考えるとモビリティに利点のあるブロードキャストによる情報の拡散が望ましいと考えられる.

ブロードキャストにより情報の拡散を扱ったもの研究の中でも緊急度を考慮したもの[3-5]や，データ特性から拡散する方向を考慮したもの[7]，マルチチャネルを利用したもの[8]などがあるがこれらの研究の共通の問題点として，遠方まで情報を伝達していく上でどの車両を中継するかという課題がある.

本章では移動体アドホックネットワークにおける中継ノード（車両）の選択方式に関して研究している論文[12],[14]を以下に紹介する.

3.2. 既存の中継ノード選択方式

ここでは The Broadcast Storm Problem in a Mobile Ad Hoc Network[12]で述べられている Flooding の問題点と様々な中継ノードの選択方式を比較評価について解説する.

3.2.1. Flooding の問題点

Flooding とはブロードキャストによりメッセージを受け取ったノードはそのメッセージが既に受け取ったメッセージかどうかに関わらず周囲のノードに再ブロードキャストを行う方式である.そのため，ノードの数だけ同様の通信が行われるため帯域が圧迫され通信が不可能になってしまうブロードキャストストームと呼ばれる問題が発生する.

3.2.2. Probabilistic Scheme

Probabilistic Scheme は再ブロードキャストの回数を低減するための最も簡易な方法であり，確率的に再ブロードキャストを行うかを決定する方式である.ブロードキャストによるメッセージを最初に受信したときに確立 P によって再ブロードキャストを行うか判定する.P=1 の場合，Flooding と同様の振る舞いになる.また，再ブロードキャストする際にラン

ダムなディレイを入れることで再ブロードキャストのタイミングを他のノードとずらすことで通信の衝突を抑制する.

3.2.3. Counter-Based Scheme

再ブロードキャストを行う場合に，メッセージの送信を開始する前に他のノードから同じメッセージを C 回受信すると，そのメッセージの送信をキャンセルする方式である.C は閾値によって決められる.この方式の処理手順を以下に示す.

① ブロードキャストされたメッセージを始めて受信したときに c=1 と初期化する.手順②において同様のメッセージを受信した場合，手順④に移行する.
② ランダムに設定されたスロットを待機する.
③ メッセージを実際に送信する.
④ 受信した車両は c の値を 1 増やし，閾値 C 以下であれば②へ.そうでなければ⑤へ移行する.
⑤ メッセージの送信をキャンセルする.

3.2.4. Distance-Based Scheme

ノードがメッセージをブロードキャストまたは再ブロードキャストするたびに，ヘッダに自分の位置を追加し，そのメッセージを受信したノードは相手との距離を計算し，閾値以下なら送信しない.そうでなければランダムな待ち時間を設定し，待ち時間終了後，同じ情報を受信していなければ再ブロードキャストを行う.

3.2.5. 評価結果

The Broadcast Storm Problem in a Mobile Ad Hoc Network[12]では上記の 3 つの中継ノード選択方式に対してシミュレーションを行っている. シミュレーションエリアには 100 ノード存在し，1 ユニット 500m のマップを 1x1，3x3，5x5，7x7，10x10 で組み合わせる. 各ノードは平均 1 秒に 1 回ランダムなタイミングでブロードキャストを行う.評価項目に関しては以下の 3 つとなる.

- Reachability(RE): ソースノードから直接か間接かを問わずブロードキャストによってメッセージを何回受信したかを示す.
- Saved ReBroadcast(SRB): (r-t)/r で表される.r はブロードキャストメッセージを受信した回数を，また t はメッセージを送信した回数を示す.
- AverageLatency:最後に再ブロードキャストされた時間から次にブロードキャストされるまでの間隔を表す.

図 5 に Probabilitistic-Scheme，図 6 に CounterBased-Scheme，図 7 に DistanceBased-Scheme の結果を示す.図 5 の P は中継するか否かの確率を示しており 20%の確率で再ブロードキャストした場合では拡散率が大きく低下することがわかる.図 6 の C は同一メッセージの何回受信するまで，再ブロードキャストを行うか閾値を表しており C=2 では拡散率が低くなることがわかる.図 7 の D は距離がどの程度離れていれば再ブロードキャストを行うかを示す.Probabilitistic-Scheme および CounterBased-Scheme と比較しても高い拡散率を示すが，再ブロードキャストの削減回数は少ない.

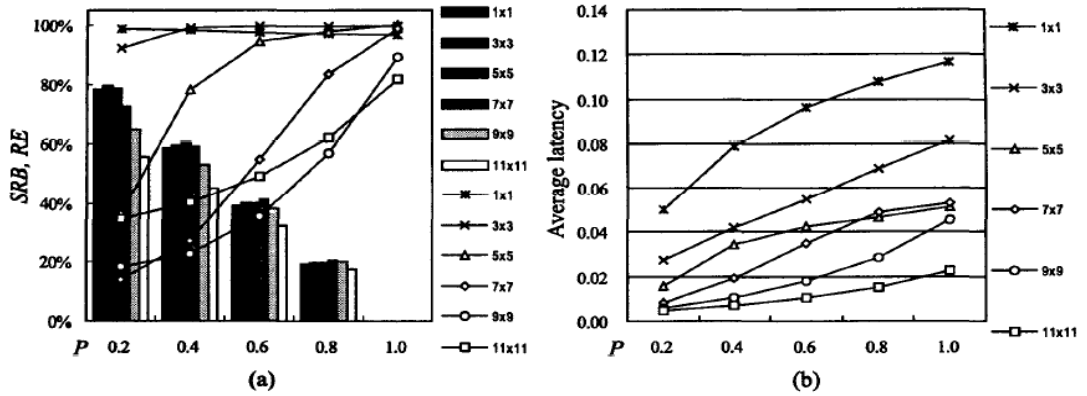

図 5　Probabilitistic-scheme の結果

引用: The Broadcast Storm Problem in a Mobile Ad Hoc Network

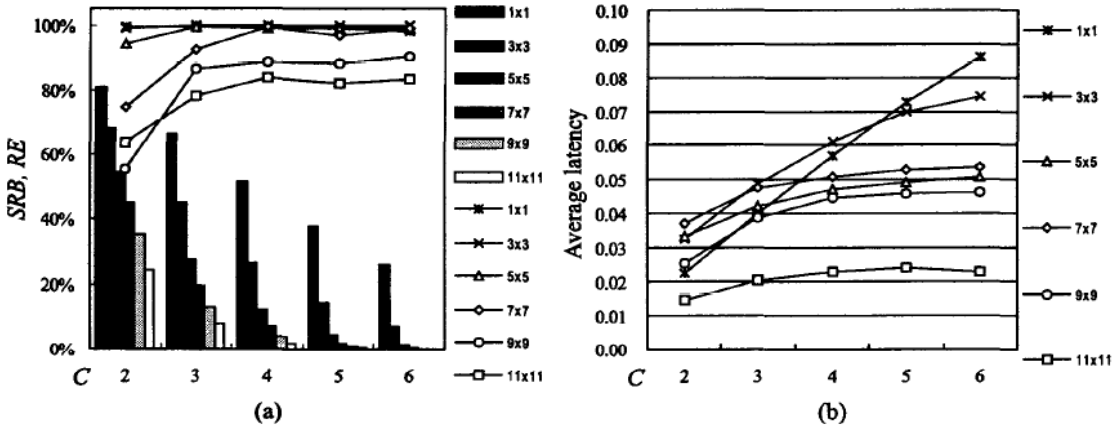

図 6　CounterBased-Scheme の結果

引用: The Broadcast Storm Problem in a Mobile Ad Hoc Network

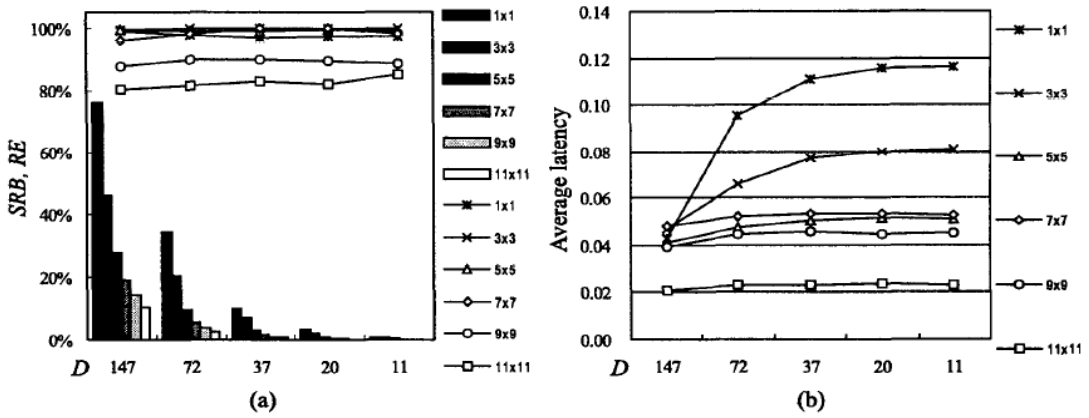

図 7　DistanceBased-Scheme の結果

引用: The Broadcast Storm Problem in a Mobile Ad Hoc Network

3.3. 都市の交通流を考慮した中継車両選択方式

A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication[14]で紹介されている距離ベースのマルチホップブロードキャスト方式である Basic Distance-Based Broadcast の説明とそれを拡張した Distance-based Multi-hop Broadcast Scheme の評価を以下に述べる.

3.3.1. Basic Distance-Based Broadcast

VANET における距離ベースのマルチホップブロードキャストの中継車両選択方式では, メッセージを受信してから中継するまでに受信者と送信者の距離に応じて以下のような式で求められる待機時間を設定する.

$$\text{WaitingTime} = -\frac{\text{MaxWT}}{\text{Range}} * d + MaxWT \qquad (1)$$

MaxWT: maximum waiting time,　Range: radio range,　d: distance between vehicles

上記の式によって決定される送信待機時間は以下の図 8 に示すように線形的な値となる.

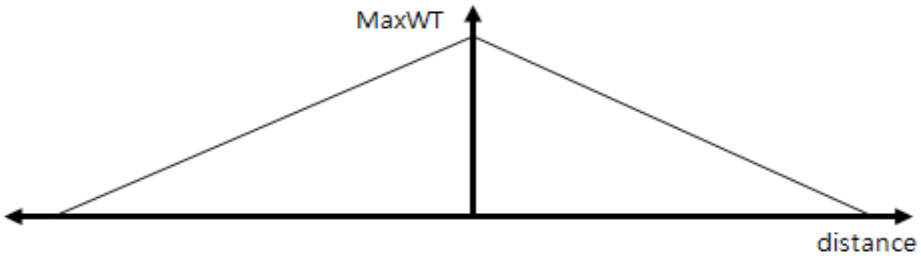

図 8　Basic Distance-Based Broadcast の送信待機時間

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

この方式の処理フローを図 9 に示す.ブロードキャストされたパケットを受け取った車両は式(1)により求められる送信待機時間を設定する.その送信待機時間中に他の車両による同一情報の再ブロードキャストを認知した場合，送信をキャンセルする.送信待機時間を経過後，同一情報の再ブロードキャストが他の車両により行われたことが認知できなければ，どの周辺車両も受信できなかったと解釈し，再ブロードキャストを行う.

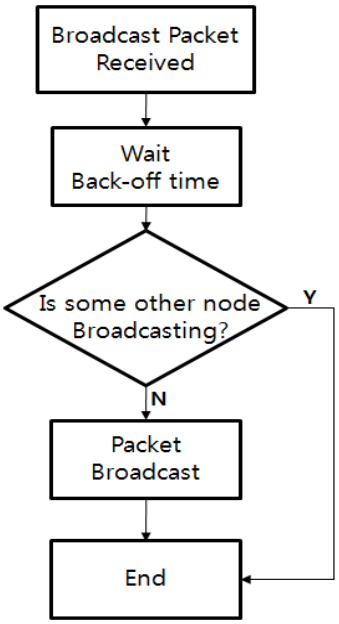

図 9 Basic Distance-Based Broadcast の処理フロー

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

3.3.2. Distance-based Multi-hop Broadcast Scheme

Basic Distance-Based Broadcast に拡張を加えて，交差点車両の考慮と距離による指数的な待ち時間を適用することでパケットの遅延を低減し，受信成功率を向上させる方式である.交差点に存在する車両の中継までの最大待ち時間は以下で設定される.

$$\mathrm{MaxWT}_{junction} = \frac{MaxWT}{Range} * R \tag{2}$$

R:交差点の直径

最大待ち時間から自車両の待ち時間を以下の式で計算する.

$$\mathrm{T} = \mathrm{x_{vhc}} * \left(\frac{MaxWT_{junction}}{R}\right) \tag{3}$$

x_vhc:交差点の中心までの距離

交差点外に存在する車両は，以下の式によって待ち時間が決定される.

$$\mathrm{EWT} = \alpha * \mathrm{e}^{-\mathrm{px}} + T_{max} \tag{4}$$

上記の式(2)，(3)，(4)により設定される各車両の送信待機時間は以下の図 10 のようになる.

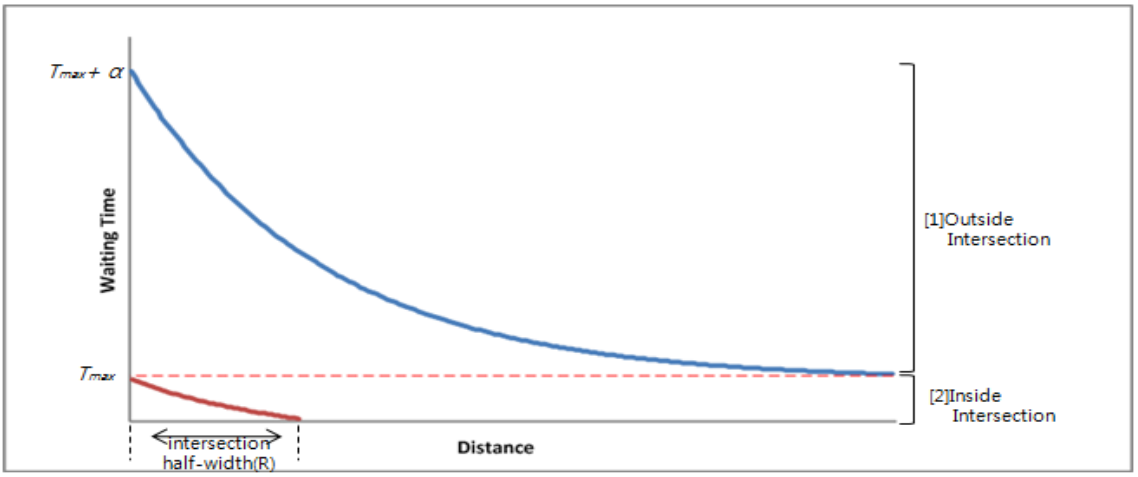

図 10　Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication での待機時間

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

3.3.3. 評価結果

A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication[14]では性能評価をネットワークシミュレータ Qualnet4.5[15]を利用して

行っている．シミュレーションエリアはマンハッタンモデルと呼ばれる[16][17]都市部の交通を模した環境に設定し，SUMO(Simulation of urban mobility)[18]によって交通流を再現する.図 11 および図 12 には線形待機時間と指数待機時間による遅延時間と受信パケット数の比較を示す.どちらの評価でも指数待機時間を用いたほうが良い結果となっている.図 13，図 14 では EWT を求める式での p の値を変化させた時の結果の違いを示す.P=0.01 のとき最も遅延時間が長く p=0.03 の時に遅延時間が最も短い.また，p=0.03 の時にパケット受信数が一番少ないことから遅延時間とパケット受信数はトレードオフの関係にあると言える.

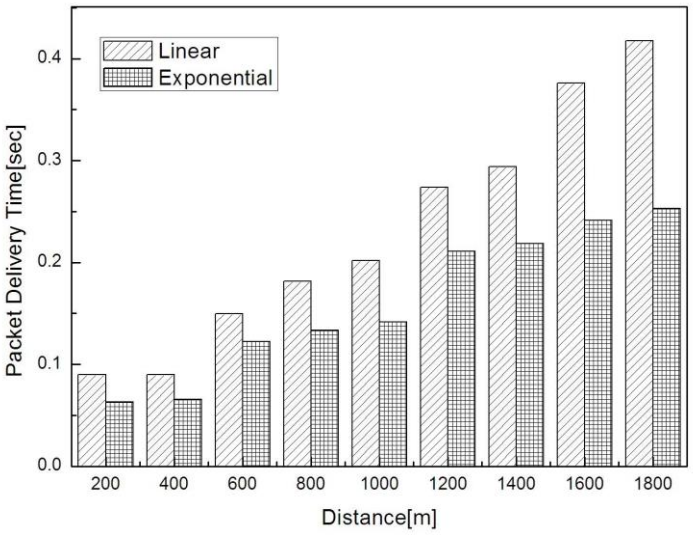

図 11 線形待機時間と指数待機時間による遅延時間比較

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

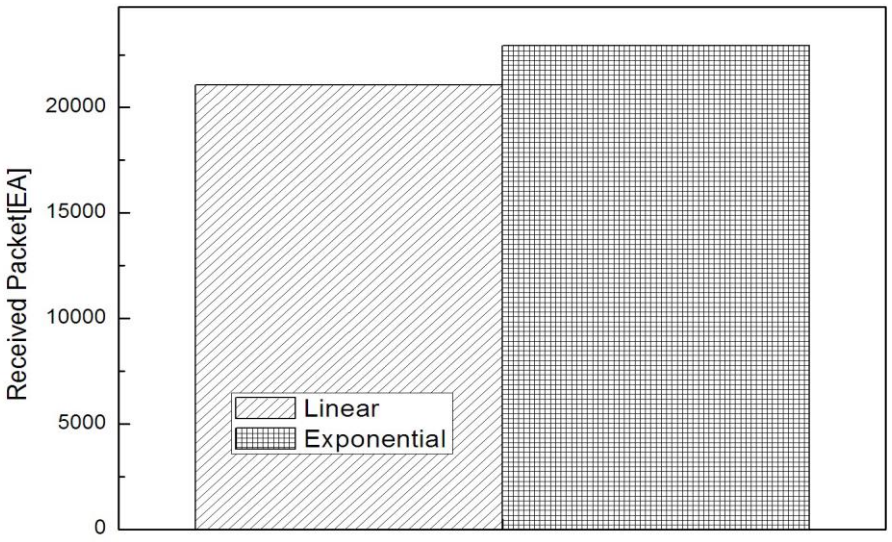

図 12 線形待機時間と指数待機時間による受信パケット数比較

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

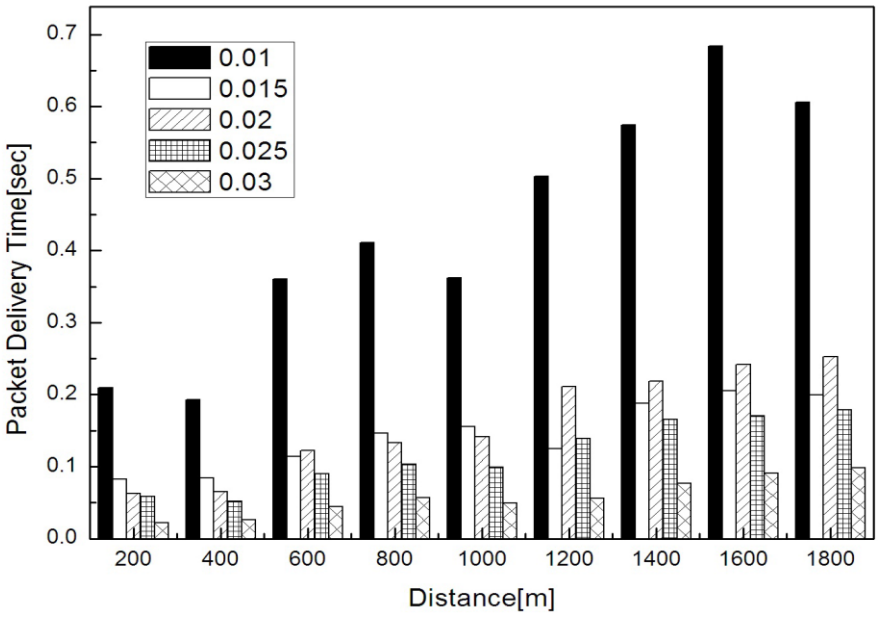

図 13　p を変化させた時の遅延時間の違い

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

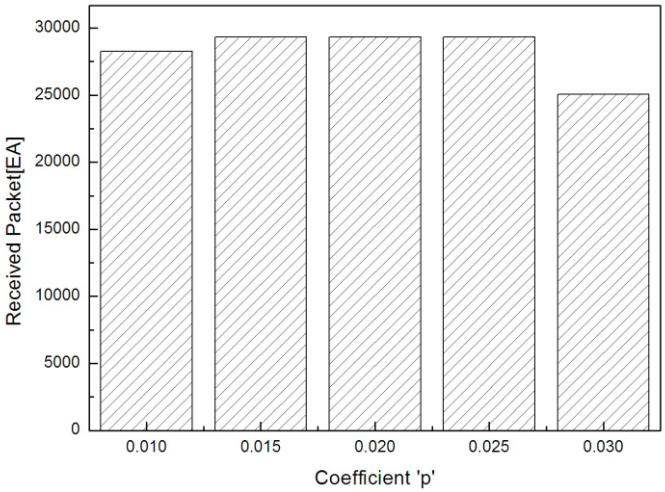

図 14　p を変化させた時のパケット受信数の違い

引用: A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication

3.4. 先行研究の問題点

以上，アドホックネットワークにおいて提案されている中継ノードの選択方式および車車間通信でのマルチホップブロードキャスト方式を紹介したが，Probabilistic Scheme，Counter-Based Scheme，Distance-Based Scheme では中継ノード(車両)の選択にランダム要素があるため必ずしも中継に最も遠いノード(車両)が選択されるとは限らないためパケットを拡散して行く上でホップ数が多くなり，冗長な通信が行われる可能性が高い.また，ホップ数を少なくするために送信者から遠方の車両を中継車両とする Basic Distance-Based Broadcast，Distance-Based Multi-Hop Broadcast Scheme For Inter-Vehicle Communication においてもほぼ同位置に存在する複数の車両が中継する場合に通信が衝突するという問題を解決できていない.

本論文ではその点に着目し，同位置に存在する車両においても効率的な中継を可能にする提案を行う.また先行研究で紹介した Basic Distance-Based Broadcast は拡散率の評価が行われていないため，それを明らかにするとともに，Counter-Based Scheme も交えて提案方式との比較評価を行う.

4. 狭域交通情報共有方式の提案

4.1. 提案の目的

2.3 項で説明した交通情報共有システムにおける課題のうちの一つとして挙げられる情報の効率的な拡散を実現するために狭域交通情報共有方式を提案する.情報の効率的な拡散とは，配布する情報が多い場合には，通信のトラフィックが圧迫され情報の拡散が適切に行われないことが想定されるため，各車両は情報を迅速に拡散させると共に通信のトラフィックを抑制することである.3 章で挙げた先行研究では，情報の中継車両選択方式としてほぼ同位置の車両が複数存在する場合には通信が衝突するという問題点があったが，本提案では位置情報を利用した明確な順序付けによる中継車両選択方式によりその問題点を解決し，拡散率の向上と遅延時間の低減を実現する.

4.2. 前提条件

本方式では，インフラ整備を必要としないというコスト面や中央集権的な処理を必要としないリアルタイム性という観点から車車間通信のみを用いる交通情報共有方式を想定する.それらに加えて日本国内での利用が目指されている，700MHz 帯と 802.11p 準拠の 5.8GHz 帯の周波数帯域を使用したブロードキャスト通信の使用を前提とする.それらの利用方法として， 安全運転支援システムでの 700MHz 帯による周囲の車両の位置，速度，移動方向を相互に通信する仕組みにより主に車両間の位置情報を共有することを想定し，本論文が提案の対象とする狭域交通情報は 5.8GHz 帯にて通信するものとする.

4.3. 機能詳細

4.3.1. 相互の車両位置把握機能

700MHz 帯を利用して，100ms 間隔で周囲の車両と位置情報および車両 ID を交換する.それを元に各車両は周囲の車両の位置を格納する.

車両位置を相互に交換する場合，以下の表 2 の情報が含まれるパケットを想定する.

表 2 位置情報パケットの構造

項目	説明
NodeId	車両番号
NodePositionX	車両のX座標
NodePositionY	車両のY座標
CurrentTime	現在時刻

4.3.2. 中継車両選択

各車両は前述の相互の車両位置把握機能で格納した位置情報を使用し，送信者の位置と自車両の位置から中継優先度を決定する，なお中継優先度は 1 が最も高いものとする．更なる中継が必要な情報を受け取った車両は受信したパケットから送信者の位置情報を取得し，自車両との距離を計算する.その後，同一の情報を受け取ったであろうと思われる車両と送信者の距離を算出し，自車両と送信者からの距離よりも遠ければ自車両の中継優先度を下げる.パケットを受信したすべての車両が同一の処理をすることにより周辺車両間で正しく中継優先度を決定できるようにする.優先度の最大値(最も優先されない値)は PRIORITY_MAX として動的に変更する．PRIORITY_MAX の値以上に中継優先度が割り当てられることはなく，中継優先度が割り当てられなかった車両が中継を行うことはない.この仕組みにより，図 15 に示すとおり受信者と送信者の距離から絶対的な中継優先度が算出する既存方式[14]とは異なり，図 16 で示すように送信元車両からの相対的な順番により決定する.算出する中継優先度に応じて，各車両の送信待機時間を設定することで中継機会に差をつける.中継優先度は送信者から最も遠方にある車両が最も高く，ほぼ同地点に他の車両がいる場合は車両 ID を比較し小さい車両を優先することにより，従来方式では通信が衝突していた問題を回避することができる.また，各車両は，自車両の送信待機状態に入っている情報と同一の情報を受信した場合，送信をキャンセルすることで冗長パケットの伝搬を抑制する.送信待機時間の詳細は後述する.

また，位置情報は 700Mhz で交換されるため交通情報を共有する 5.8GHz よりも遠くまで電波が到達することを想定すると，各車両が取得した位置情報に基づきすべての車両に中継優先度を付与する場合，実際の交通情報の通信では電波が到達しない可能性がある.そのため，想定される 5.8GHz での通信可能距離内(380m と設定)でのみ中継優先度を付与することとする.また，距離上では通信可能と判断される場合でも電波遮蔽物がある場合には，

電波が到達しない可能性を加味し，中継優先度を付与しない.電波の遮蔽の予測は，中継車両と送信元車両の x 軸と y 軸の座標がともに大きく異なる場合は道路の筋や通りが異なっている可能性が高いと判断することで実現する.

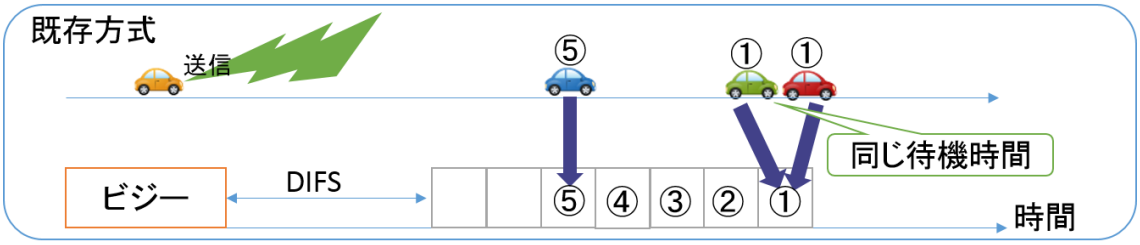

図 15 距離情報を用いた既存の中継車両選択方式

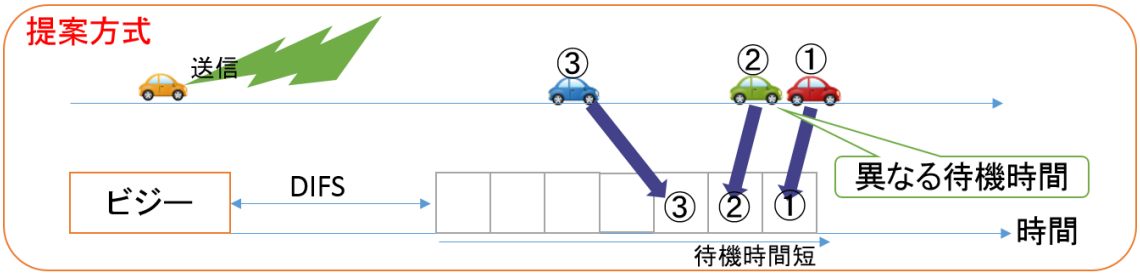

図 16 位置情報を利用した順序付けによる中継車両選択方式（提案方式）

- 中継優先度の割当て車両数(PRIORITY_MAX)の動的変更

  中継優先度を設定し，中継が可能となる車両の台数を PRIORITY_MAX として表現する.中継優先度は，情報を受信した全ての車両に付与するわけではなく，通信状況によって変化する PRIORITY_MAX の値の台数に付与される.PRIORITY_MAX の値が大きければ，仮にいくつかの中継車両におけるパケットが衝突したとしても多くの優先度の低い車両が中継することで情報の喪失を防ぐことができる．一方で，トラフィックが混雑した場合では，冗長な通信により輻輳が発生しやすくなる傾向があるため，通信のトラフィックの混雑度に応じて PRIORITY_MAX の値を増減させることで効率的な情報の拡散を実現する.PRIORITY_MAX の違いによる中継優先度の例を以下の図 17 に示す.

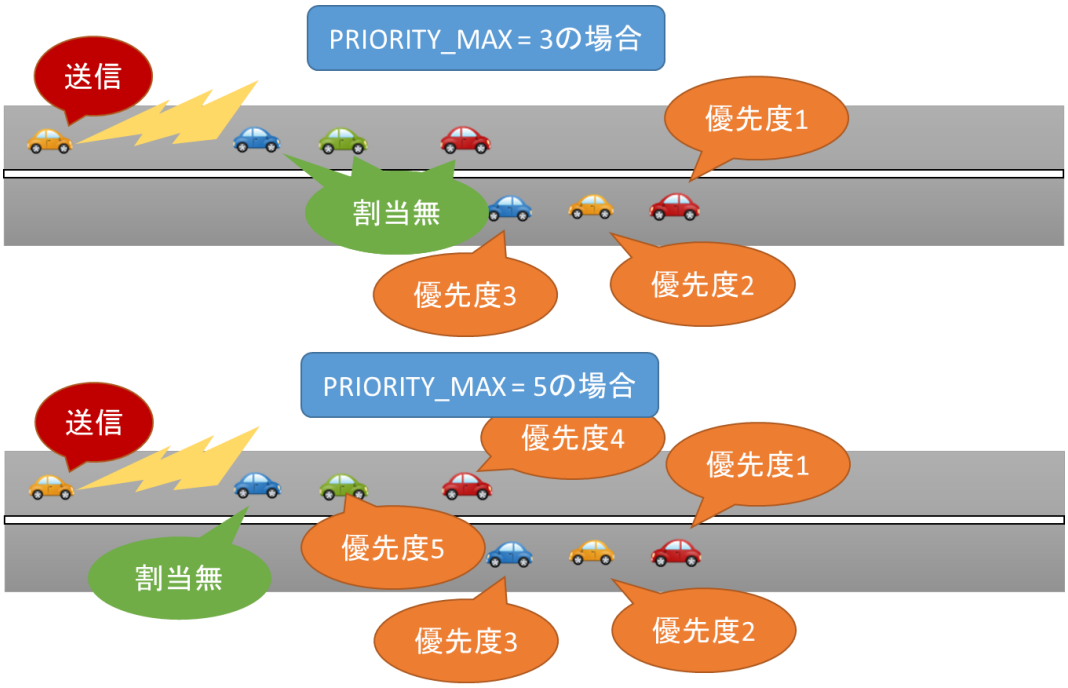

図 17 PRIORITY_MAX と中継優先度の割当例

- 交差点存在車両に対する特別優先度の割当て

　交差点とは道路と道路が交わる点の 30m 以内と本論では定義する.交差点での中継は多方向（例えば 4 方向）への情報の拡散が可能になるため，特別な中継優先度として交差点優先度を 20 台まで付与する.交差点車両への交差点優先度は前述した通常の割当てによる中継優先度よりも大きい値，つまり低優先度に設定する.あらかじめ地図データから交差点の座標を取得しておき，その近傍に存在する車両は交差点車両として交差点中継優先度を設定する.通常の優先度と交差点優先度の割当区分を以下の表 3 に示す.

表 3 中継優先度の割当区分

優先度種類	最小値	最大値
中継優先度	1	PRIORITY_MAX
交差点優先度	PRIORITY_MAX + 1	PRIORITY_MAX + 20

- 送信待機時間の詳細

　中継優先度により算出する送信待機時間は MAC 層に実装する.アプリケーション層にて決定した送信待機時間は MAC 層でのバックオフ時間に反映する.表 4 に示す通り，バックオフ時間とは，通常 SIFS(Short Interframe Space)と呼ばれる固定長の待機時

間とランダムな値×スロットタイムにて決定される可変長の待ち時間（バックオフ時間）であるが，提案方式では中継優先度×スロットとする．

表 4 既存方式と提案方式の MAC 層待機時間の違い

	MAC層待機時間
既存手法	SIFS + SlotTime * Random(0 , ContentionWindowSize)
提案手法	SIFS + SlotTime * MyPriority(中継優先度)

以上の各機能の処理を踏まえて各車両における中継処理のシーケンスを図 18 に示した．また図 19 に中継優先度の割当て例を示す.ここでは, すべての車両は送信車両からの受信範囲内にいるとする.中継優先度は値が小さいほど優先するため，最も遠方の車両から中継優先度 1 が割り当てられる.一方，交差点の近傍にいる車両は通常の中継優先度とは異なる交差点優先度を割当てる．

中継車両に選択された車両はパケットを送信するが，パケットは以下の表 5 に示す情報が含まれるものを想定する．

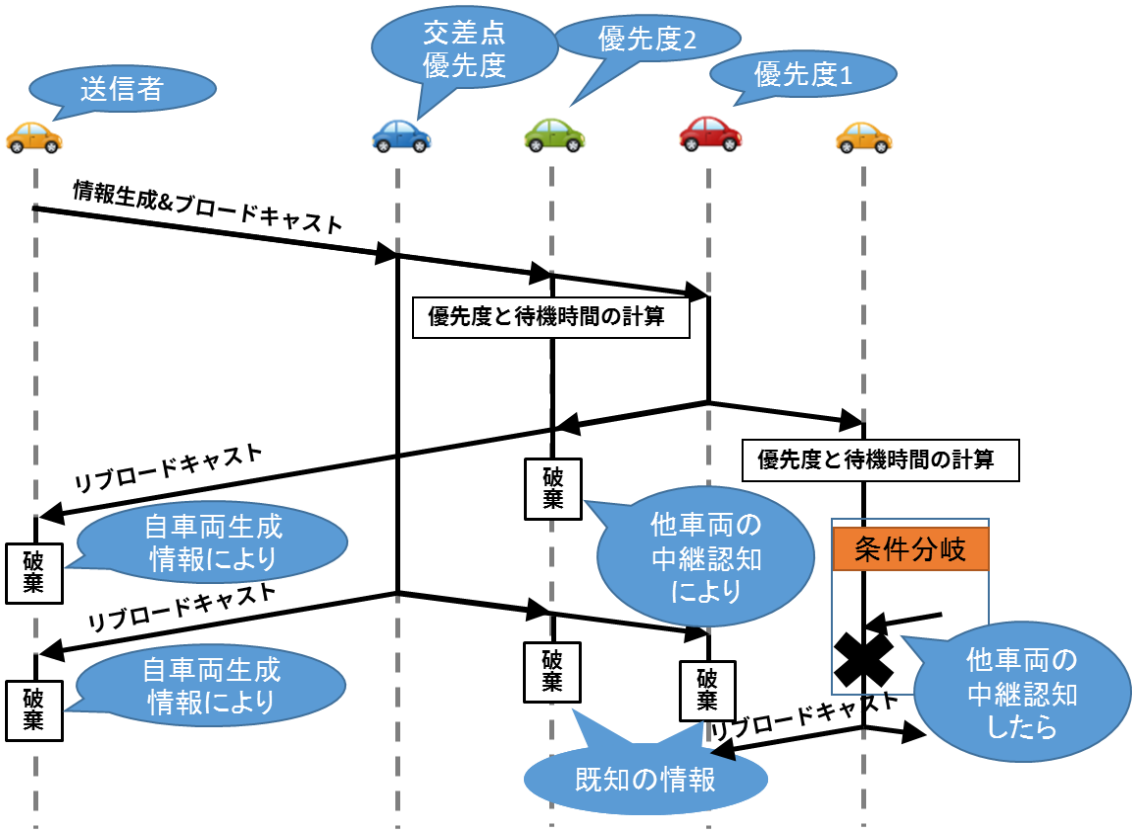

図 18　中継処理のシーケンス図

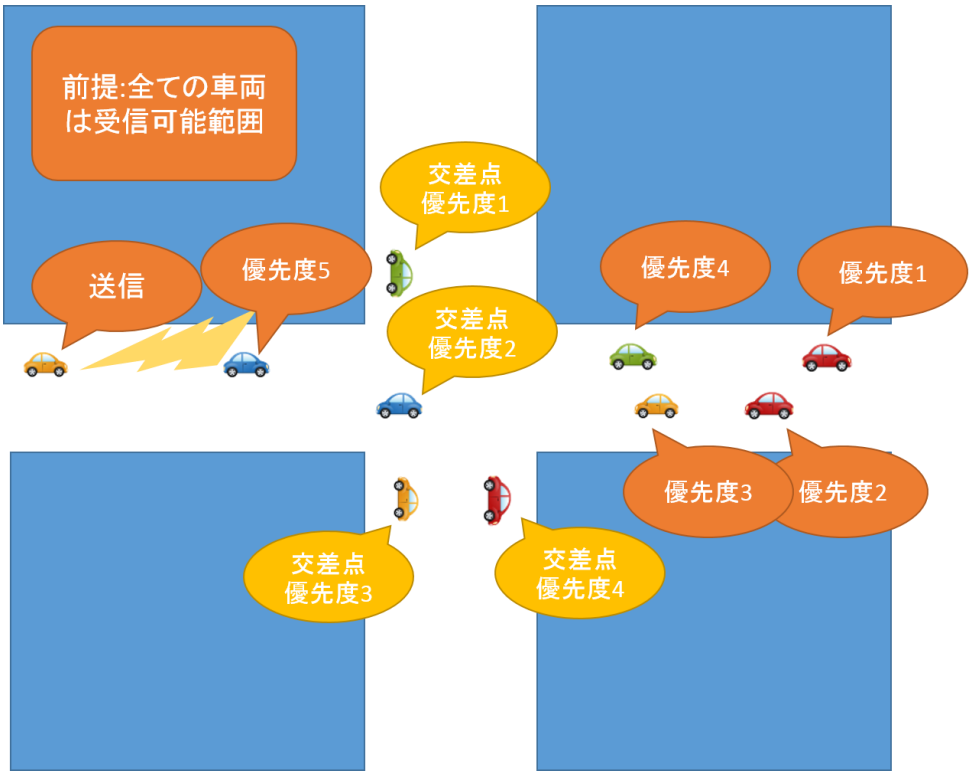

図 19 中継優先度の割当例

表 5 パケットの構造

項目	説明
BroadcastTime	情報を生成した時間
SourceNodeId	情報を生成した車両番号
SourceNodePositionX	情報を生成した車両のX座標
SourceNodePositionY	情報を生成した車両のY座標
maxHopCount	最大ホップ数
counterThreshold	中継回数の閾値
distanceThresholdInMeters	距離により中継するかの閾値
RelayPriority	中継優先度
RelayNodeId	中継車両の車両番号

4.3.3. 各機能のクロスレイヤ制御

提案方式では，表 6 にようにアプリケーション層で各車両の位置情報の管理，および中継優先度の算出を行い MAC 層で適用を行うクロスレイヤ制御を行う．

表 6 レイヤごとの機能の割当て

レイヤー名	各機能の実装による追加と変更
アプリケーション層	700MHzによる相互位置取得 中継車両優先度の算出
トランスポート層	変更なし
ネットワーク層	変更なし
MAC層	中継優先度を反映したバックオフ時間
PHY層	変更なし

5. シミュレーション評価

5.1. 概要

簡易的な都市部での交通流を模したシミュレーションを行うためにネットワークシミュレータ Scenargie[19]を用いて提案方式を評価した.評価の際には提案方式の有効性を示すために比較方式として先行研究で用いられている中継車両選択方式[12,14]と比較する.既存方式の実装詳細は後述する.

5.2. Scenargie について

現在，研究分野で利用されるネットワークシミュレータにはいくつか種類があるが，特に Scenargie を提案方式のシミュレーション評価で用いた． その理由は，Scenargie では車車間通信において主要である IEEE 802.11p に詳細な点まで対応している点や, 車両を自動車の動きに模して移動させ，実在の環境により近づけるために建物を配置できる点，多くの電波伝搬モデルをサポートしている点から優れているからである.

5.3. 評価項目

本提案と既存方式のシミュレーション評価において取得した統計値を表 7 に示す.

表 7 取得する統計値

項目	説明
情報生成数	生成された情報の数
情報取得数	取得できた情報の数
拡散率	情報取得数 / 情報生成数 * 情報生成数
最高拡散率	全ての情報の中で最も高い拡散率
最低拡散率	全ての情報の中で最も低い拡散率
拡散率分布	それぞれの情報の拡散率の分布
平均遅延	情報が生成されてから車両に届くまでの平均時間
最大遅延	情報が生成されてから車両に届くまでの最大時間
遅延時間分布	それぞれの情報の遅延時間の分布
総送信回数	全ての車両の送信回数の和
中継回数	全ての車両の中継回数の和
受信成功回数	フレームの受信に成功した回数の和
フレーム破損数	フレームの受信に失敗した回数の和
エラーレート	フレーム破損数 / (受信成功回数 + フレーム破損数)

5.4. シミュレーションシナリオと設定値

シミュレーションは図 22 に示す交差点間隔 400m で 4×4 のグリッド状のマップで行う．各交差点には信号機が設置し，各ノードはランダムに決定された初期位置からランダムに移動し，一定間隔で情報を生成する.その他のシミュレーションにおける設定値は表 8 にまとめた.

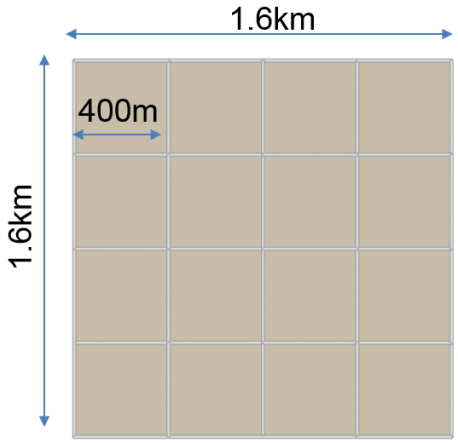

図 20 シミュレーションエリア

表 8 シミュレーションの主な設定値

項目	説明	値
通信方式	通信に利用するプロトコル	IEEE 802.11p
伝送速度	通信の速度	3Mbps
パケットサイズ	拡散させる情報の大きさ	128byte-512byte
最大ホップ数	何回まで中継を許すか	100
電波伝搬モデル	電波シミュレーションのモデル	ITU-R_P.1411
情報生成頻度	新しく情報を生成する頻度	0.5s/台-8s/台
評価エリア	シミュレーションマップ	1.6km x 1.6km(グリッド)
交差点間隔	交差点の間隔	400m
信号	信号の有無	有
車線数	道路の片側車線数	2
車両位置	開始時の車両位置	ランダム
車両台数	シミュレーションの車両台数	500台
車両長	車両の長さ	5m
車両速度	走行時の車両速度	30km/h-60km/h
実行時間	シミュレーションの時間	120s

5.5. 比較方式の実装詳細

5.5.1. Counter-Based Scheme 方式(比較方式 1)

比較方式として第 3 章で解説した Counter-Based Scheme[12]を扱う.Counter-Based Scheme における情報の拡散手順を説明する.まず各車両は情報を生成し，発信する.情報を受信した車両は，ランダムな待ち時間を設定し，情報を送信する.待ち時間はアプリケーション層と MAC 層により発生する.アプリケーション層での待ち時間は以下の式(5)により決める.

$$\mathrm{WaitTime} = \mathrm{Random}(0.5 - 0.1) * SECOND \tag{5}$$

MAC 層での待ち時間は SIFS と呼ばれる決められた値とコンテンションウィンドウサイズにより与えられる幅を持たせたランダム値×スロット数にて決める.

アプリケーション層での待ち時間中では，同一のパケットを閾値 C 回受信することで，他の車両が中継したことを判別し，同一パケットの送信をキャンセルする.
なおこの評価では，中継するか否かを判別する閾値 C=1 と設定し，一度でも同一パケットが受信された場合には送信をキャンセルすることとした.

5.5.2. Basic Distance-Based Broadcast (比較方式 2)

既存研究の Basic Distance-Based Broadcast(BDBB)で用いられている方式[14]であり，通信可能距離を任意の分割するセクタ数で割り，送信車両と受信車両の距離を計算して自車両が何番目のセクタに存在するかにより待機時間を設定する方法である.待機時間は以下の式(6)，(7)で求める.ここでは，SPLITNUM=100 ， Distance = 380 として設定した.

$$\mathrm{MySector} = \frac{\mathrm{Distance} * \mathrm{MAXRANGE}}{\mathrm{SPLITNUM}} \tag{6}$$

Distance:送信車と受信車の距離 MAXRANGE:通信可能距離 SPLITNUM:分割数

$$\mathrm{WaitTime} = \frac{(0.5 - 0.1)}{SPLITNUM} * MySector \tag{7}$$

上記の式(6), (7)から求めた待機時間は送信車と受信車の距離に応じて線形的に変化する. そのため，ほぼ同じ位置に複数の車両がある場合には同一の待機時間が設定され，パケットが衝突する可能性がある.比較方式 1 と同様で，アプリケーション層での待ち時間中に同一のパケットを受信することで送信をキャンセルすることができる.

6. 評価結果と考察

シミュレーションにより得られた結果とそれに対する考察を以下に述べる.

6.1. 拡散率評価と考察

情報生成間隔による拡散率の変化を図 21 に表す.図 21 より，全ての情報生成間隔で提案方式の拡散率が最も高く，特に情報生成間隔が 0.5s の時に比較方式 1(Counter-Based Scheme)よりも平均拡散率が最大 35.0%，比較方式 2 (BDBB)よりも最大で 1,065%向上している.今回のシミュレーションシナリオでは信号により，交差点付近に車両が多く存在するため比較方式 2(BDBB)おいて通信が衝突しやすいことが原因で比較方式 2(BDBB)が最も悪い結果になったと考えられる.多くの既存研究では比較方式 2(BDBB)が利用されている一方で，信号のある交通流でのシミュレーションが行われていなかった.この結果より，比較方式 2(BDBB)が比較方式 1(Counter-Based Scheme)よりも必ずしも効率がいいとは言えないことが明確になった.

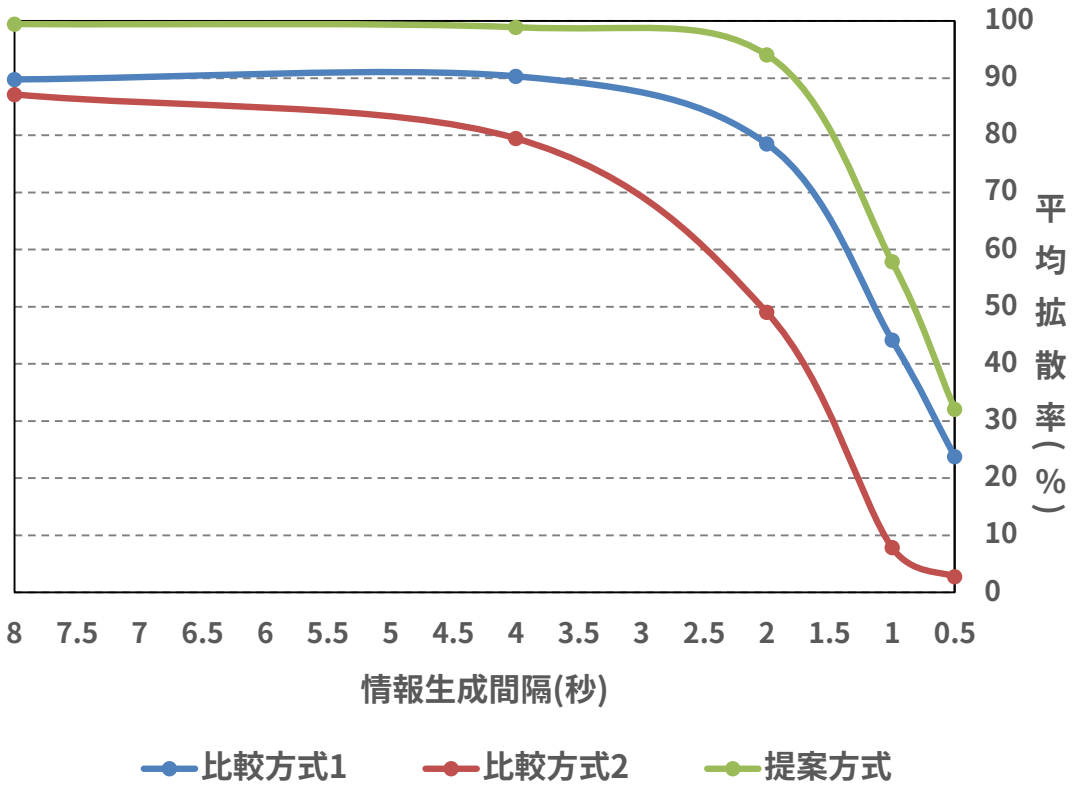

図 21 情報生成間隔と平均拡散率の変化

情報生成頻度が 8s の時の比較方式 1(Counter-Based Scheme)，比較方式 2(BDBB)と提案方式の拡散率分布を，それぞれ図 22，図 23，図 24 に示す.結果から提案方式では比較方式 1(Counter-Based Scheme)，比較方式 2(BDBB)よりも全ての分布で拡散率が向上していることが確認できる.また比較方式 1(Counter-Based Scheme)および比較方式 2(BDBB)では 100%まで達している情報は少なく，提案方式ではシミュレーションエリアの隅々の車両まで情報を拡散できていることがわかる.

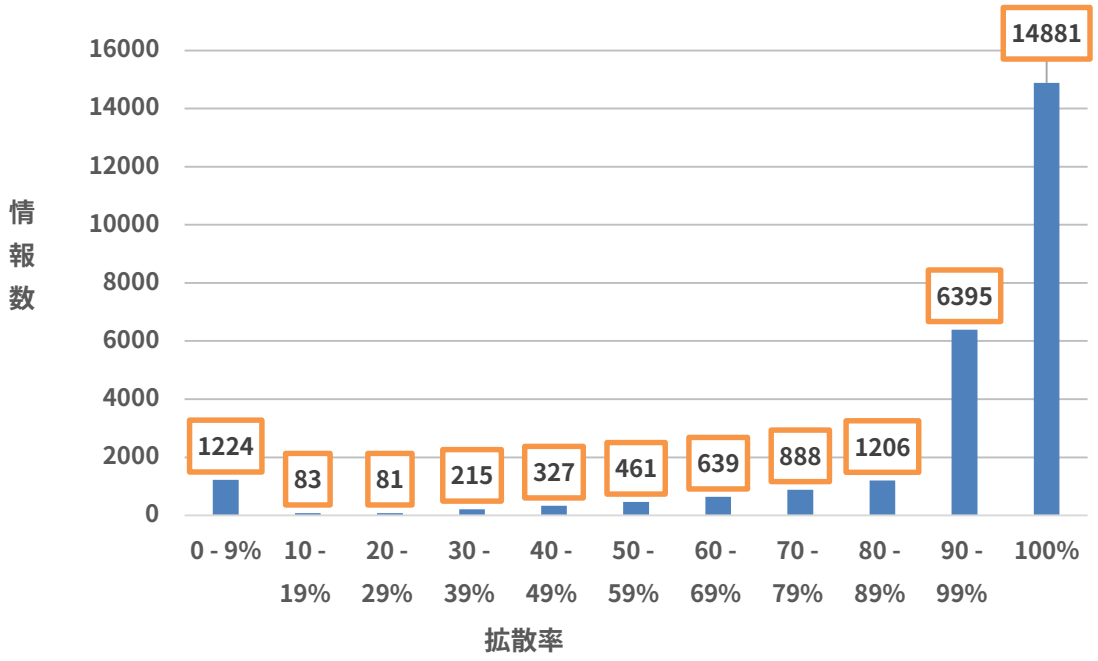

図 22 比較方式 1 の拡散率分布

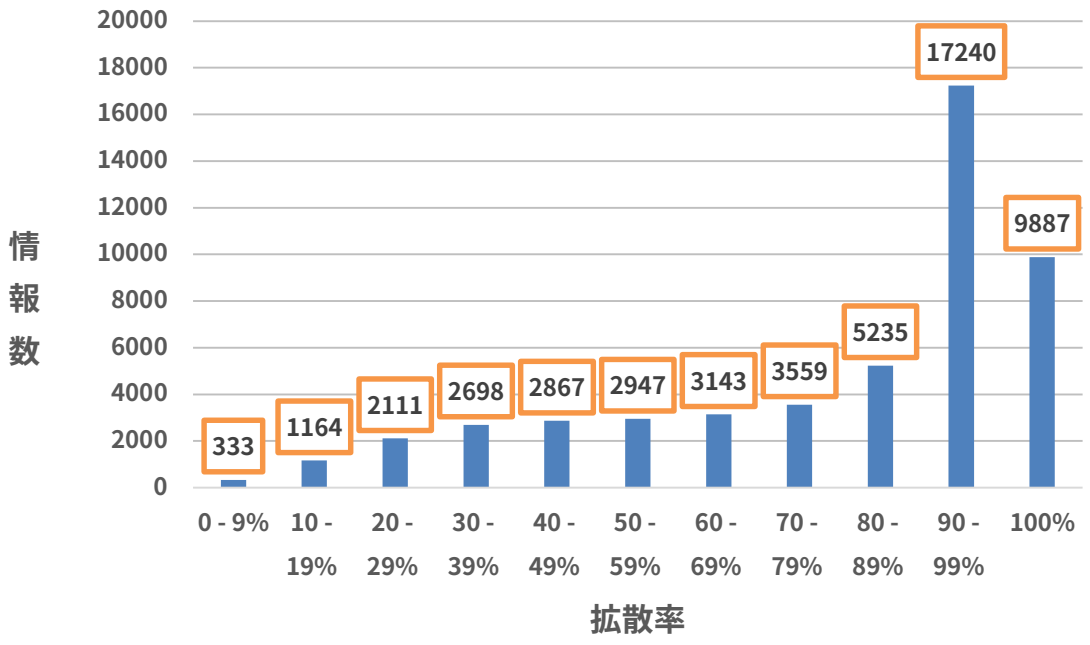

図 23 比較方式 2 の拡散率分布

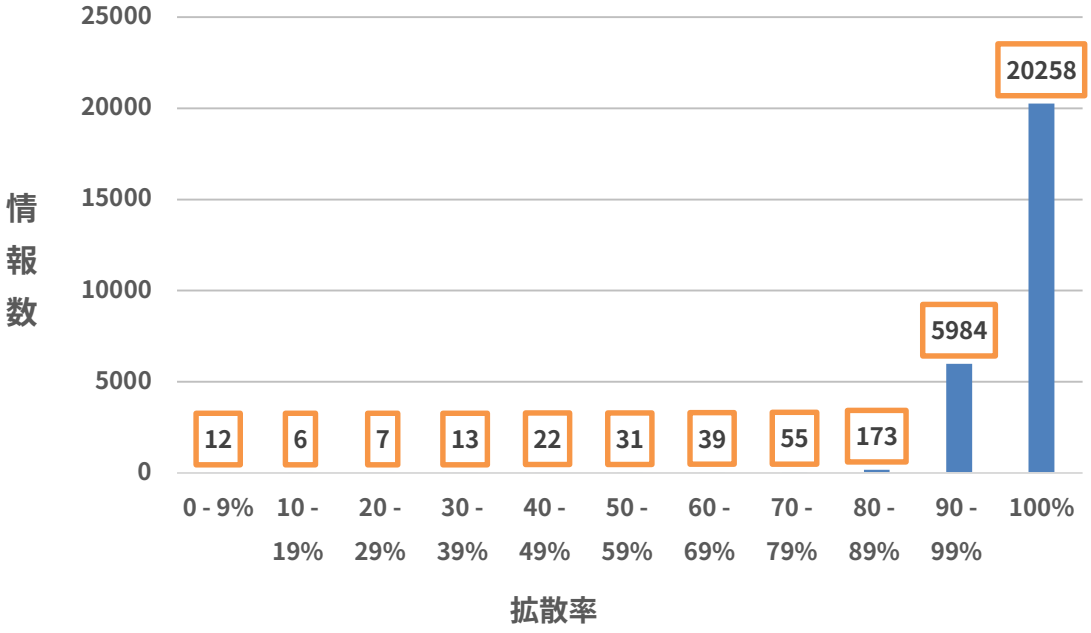

図 24 提案方式の拡散率分布

6.2. トラフィック削減評価と考察

情報生成間隔による情報送信回数を図 25 に示す.図 25 より，8s-2s では提案方式と比較方式 1(Counter-Based Scheme)の総送信回数は同等であり，1s，0.5s 間隔では総送信回数を最大 20.5%削減できていることがわかる.同様に提案方式と比較方式 2(BDBB)では全ての情報生成間隔で比較方式 2(BDBB)の送信回数が多くなり情報生成間隔が 1s の時に最大 28.25%削減できている.

また，拡散率評価の結果から，全ての状況で提案方式は比較方式 1(Counter-Based Scheme)および比較方式 2(BDBB)より高い拡散率を維持しており，一回の送信によってより多くの車両へ情報を拡散できていることがわかる.

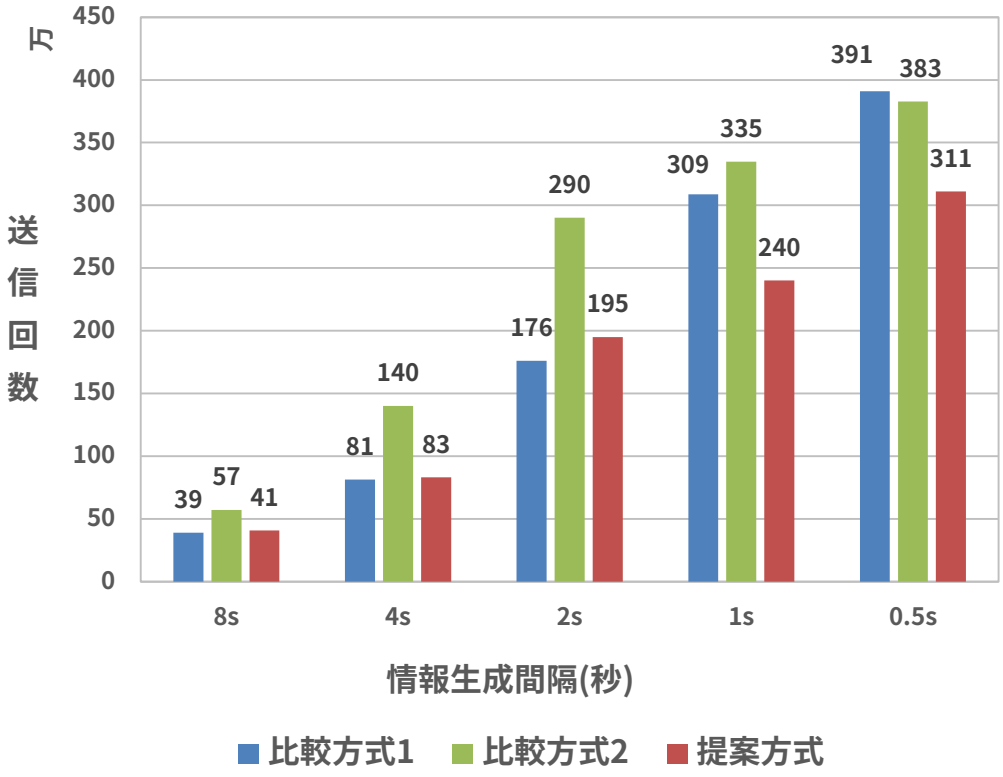

図 25　各方式の総送信回数

図 26 と図 27 に情報生成間隔ごとの受信フレーム数と破損フレーム数, 図 28 にエラーレートをそれぞれ示す.受信フレーム数のグラフより，8s から 4s，4s から 2s 間隔で情報を生

成する場合，時間当たりで倍の情報が生成されることより同等の拡散率を維持する時，受信フレーム数も約 2 倍になるはずである.しかし 2s から 1s および 1s から 0.5s では受信成功数の増加率は鈍化しており，通信トラフィックに空きがなくなっていることが予想される.また，提案方式では比較方式 1(Counter-Based Scheme)，比較方式 2 と比較して，受信成功数は同等かそれ以下にもかかわらず高い拡散率を示していることよりから，効率的に情報を拡散できていることがわかる.一方，比較方式 1(Counter-Based Scheme)，比較方式 2(BDBB)では，同一の情報を複数回受信しているため受信成功回数は多いにもかかわらず拡散率に反映されていないと考えられる.加えて図 27 および図 28 より，情報生成間隔 1s, 05s の時, 提案方式では破損フレーム数が少なくエラーレートが比較方式 1(Counter-Based Scheme)よりも低い値で抑えられており，信頼性の高い通信ができていることがわかる.一方で，8s-2s の時にエラーレートと破損フレーム数が提案方式で高い値を示しているのは，トラフィックに空きがある場合に PRIORITY_MAX を高い値に設定し情報の拡散に冗長性を持たせていることが原因と考えられる.

比較方式 2(BDBB)については，信号待ちの車両など近くの位置に複数の車両が存在する場合，通信がかならず衝突してしまうためフレーム破損数およびエラーレートが提案方式および比較方式 1(Counter-Based Scheme)よりも高い値を示しており，フレーム破損数は 1,200 万，エラーレートは 20%前後で停滞していることがわかる.

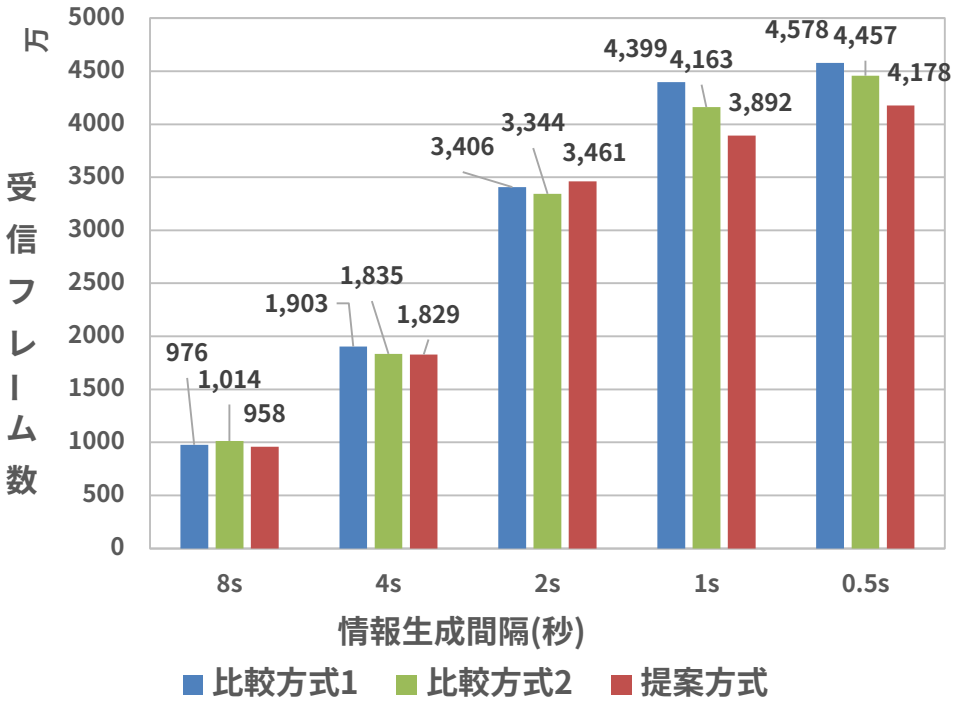

図 26 各方式の受信フレーム数

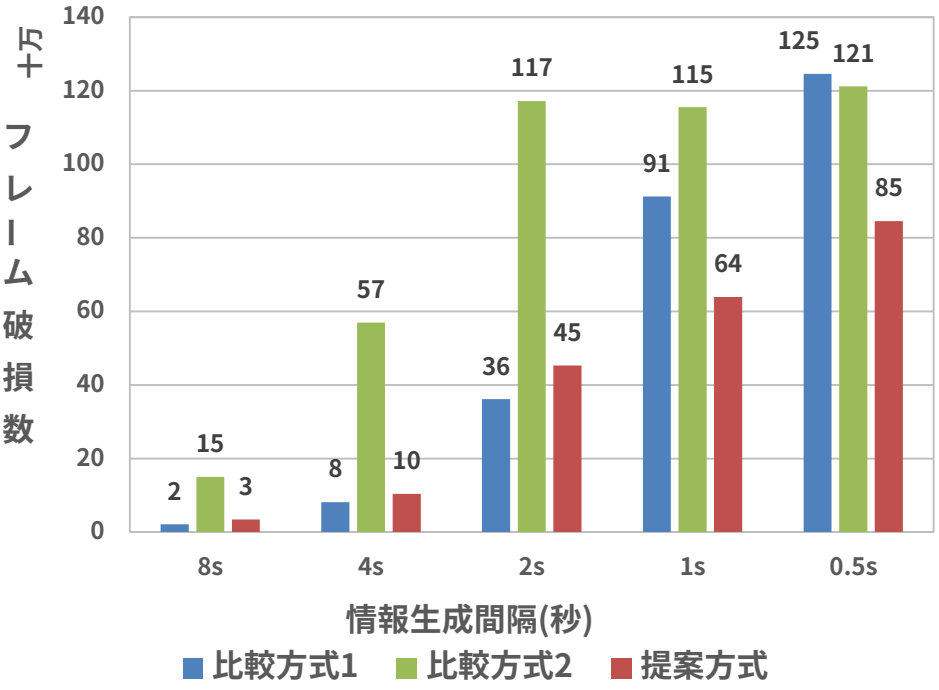

図 27 各方式の破損フレーム数

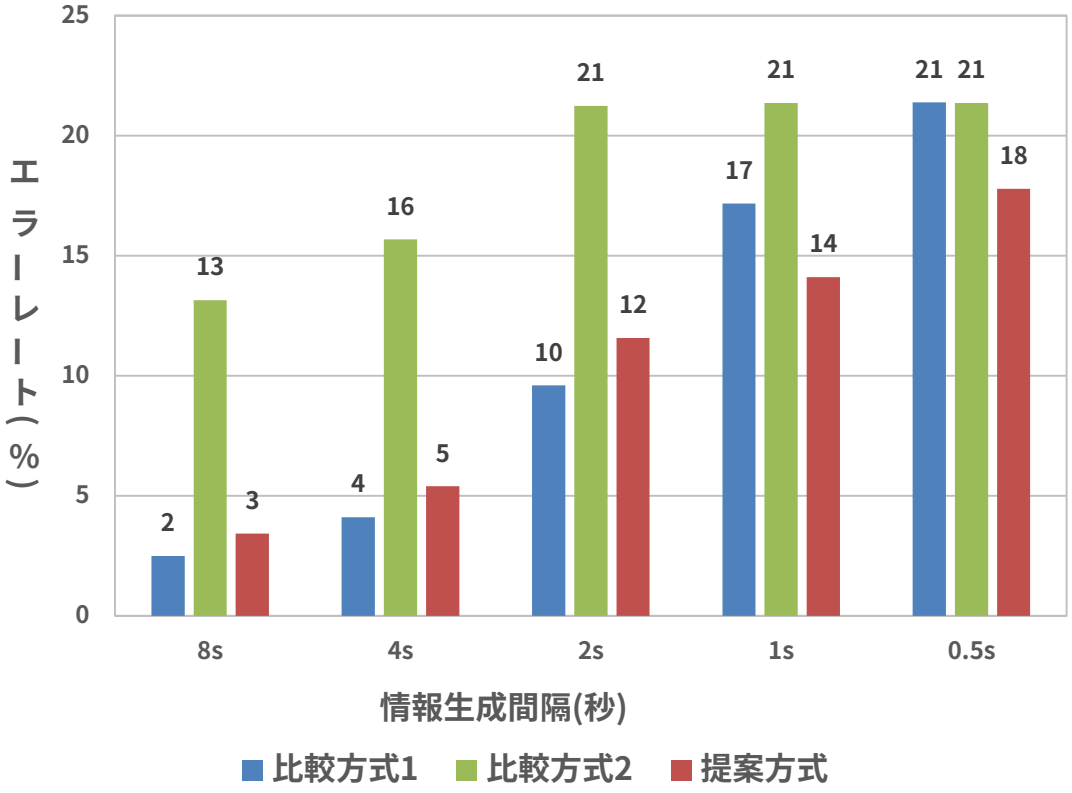

図 28 各方式のエラーレート

6.3. 遅延時間評価と考察

情報生成間隔による平均遅延時間を図 29 に示す.図 29 より，提案方式では比較方式 1(Counter-Based Scheme)よりも最大で 55.3%，比較方式 2(BDBB)よりも最大 62.9%遅延時間を低減している.提案方式では，送信者から最も遠くの車両を中継車両に選択することにより情報のホップ数を減少させる点や交差点車両を中継車両と選択することで回り込みによる遅延時間の増大を防いでいる点が大きく寄与している.また提案方式と比較方式 1(Counter-Based Scheme)において情報生成間隔が 1s の時よりも 0.5s の時に遅延時間が減少している理由として，0.5s の場合にはトラフィックが圧迫されることで中継の際にデータが失われ，ホップ数が多く必要であり遅延時間が大きくなりがちな遠くの車両には届かないことが原因と考えられる.比較方式 2(BDBB)での情報生成間隔 2s から 1s の結果でも同様の現象が起きていると考えられる.また，情報生成間隔 8s および 4s では比較方式 1(Counter-Based Scheme)より比較方式 2(BDBB)の方が平均遅延時間は小さいという結果から比較方式 2 での最遠車両の選択方法の有効性を示していると言える.

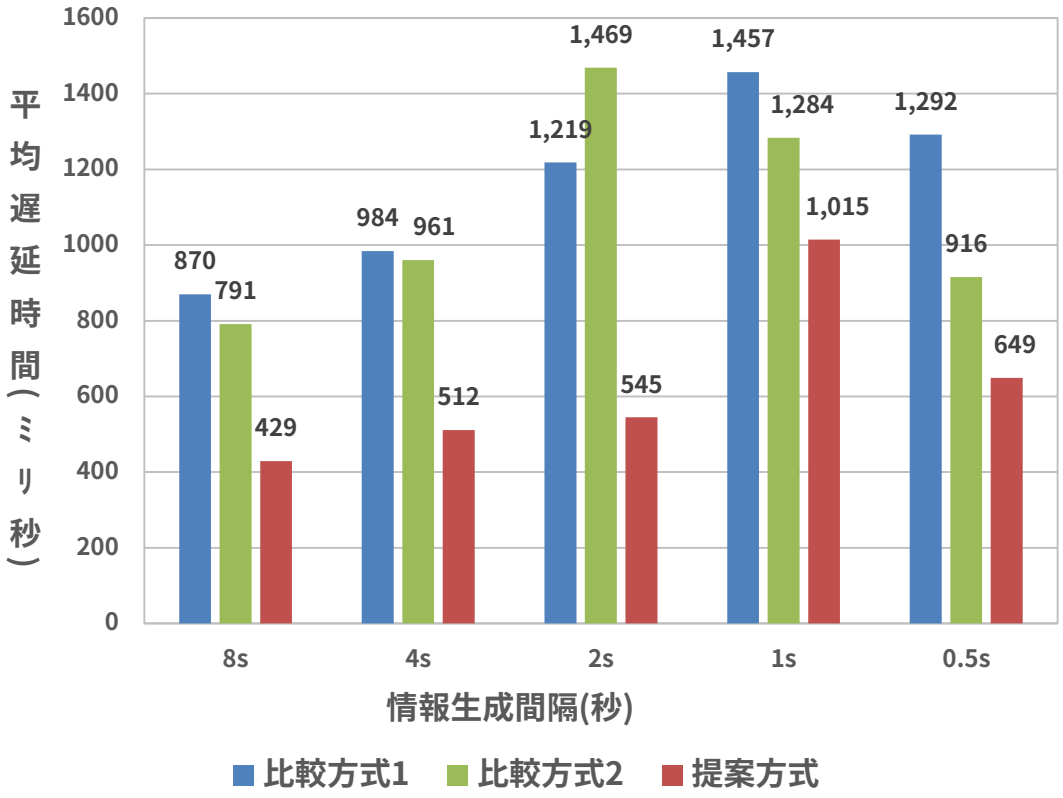

図 29 各方式の平均遅延時間

図 30，図 31 に各方式の最大遅延時間と最小遅延時間を示す.最大遅延時間では情報生成頻度 8s の時に提案方式での最大遅延時間が 23.5%減少していることがわかる.しかし，4s よりも情報生成間隔が短い場合，最大遅延時間に大きな差は見られない.また最小遅延時間ではいずれの方式でも大きな差は出ていない.

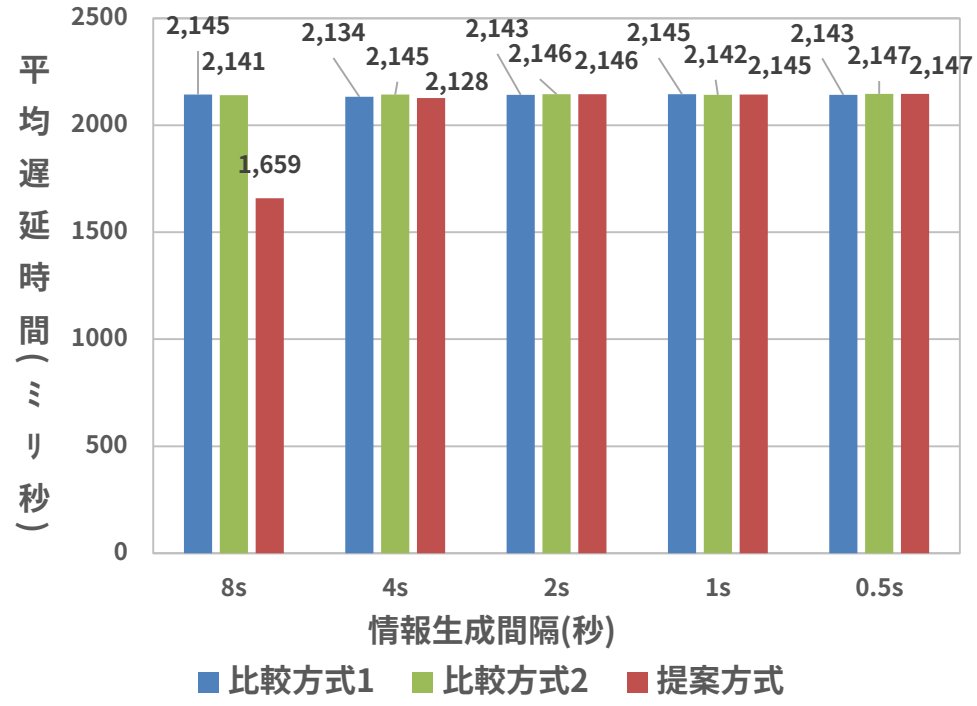

図 30 各方式の最大遅延時間

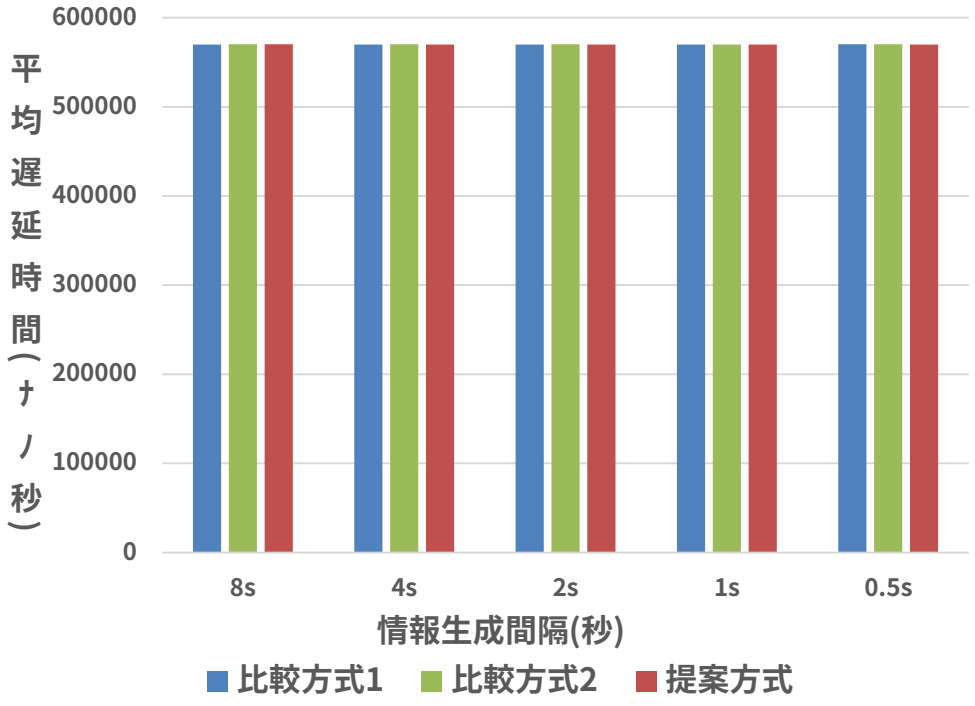

図 31 各方式の最小遅延時間

6.4. データサイズの違いによる結果と考察

データサイズを 256byte および 512byte に変化させた場合に拡散率がどのように変化するかを図 32，図 33 に示す.結果からデータサイズを大きくした場合でも，提案方式は比較方式 1(Counter-Based Scheme)および比較方式 2 よりも平均拡散率が向上していることがわかる.256byte の時，提案方式は比較方式 1(Counter-Based Scheme)より情報生成間隔 1s のとき最大で 31.7%，比較方式 2(BDBB)より情報生成間隔 0.5s のとき最大で 690%の拡散率向上が見られる.512byte では提案方式は比較方式 1(Counter-Based Scheme)よりも情報生成間隔 2s の時，最大で 36.6%，比較方式 2 よりも情報生成間隔 0.5s の時最大で 416.1%拡散率が向上していることがわかる.

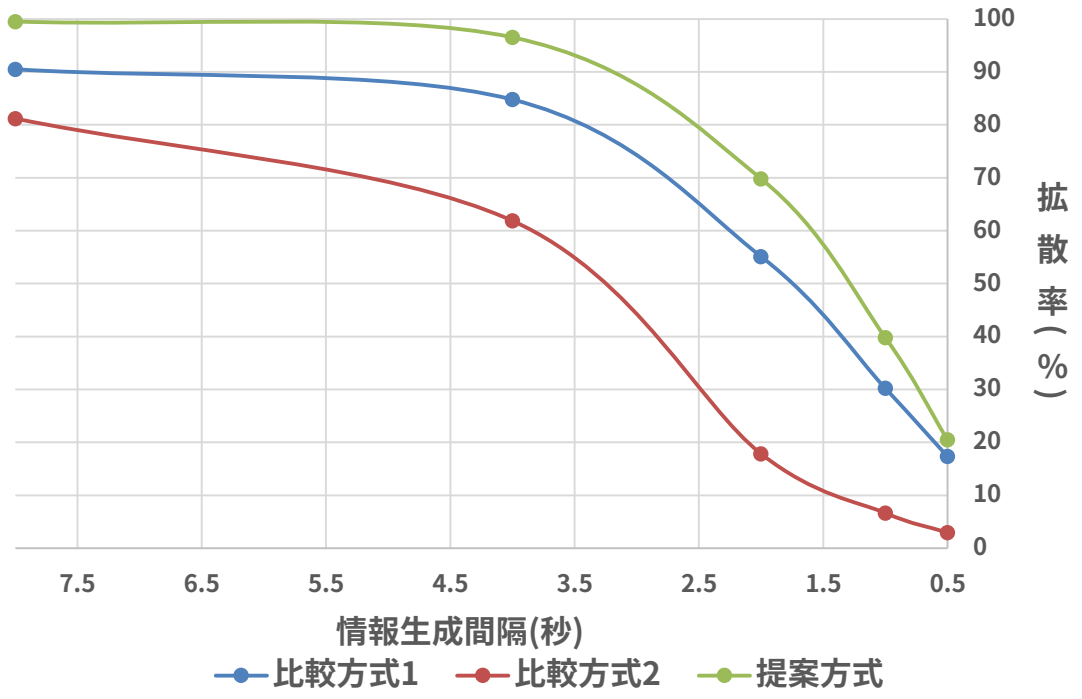

図 32 データサイズ 256byte における拡散率

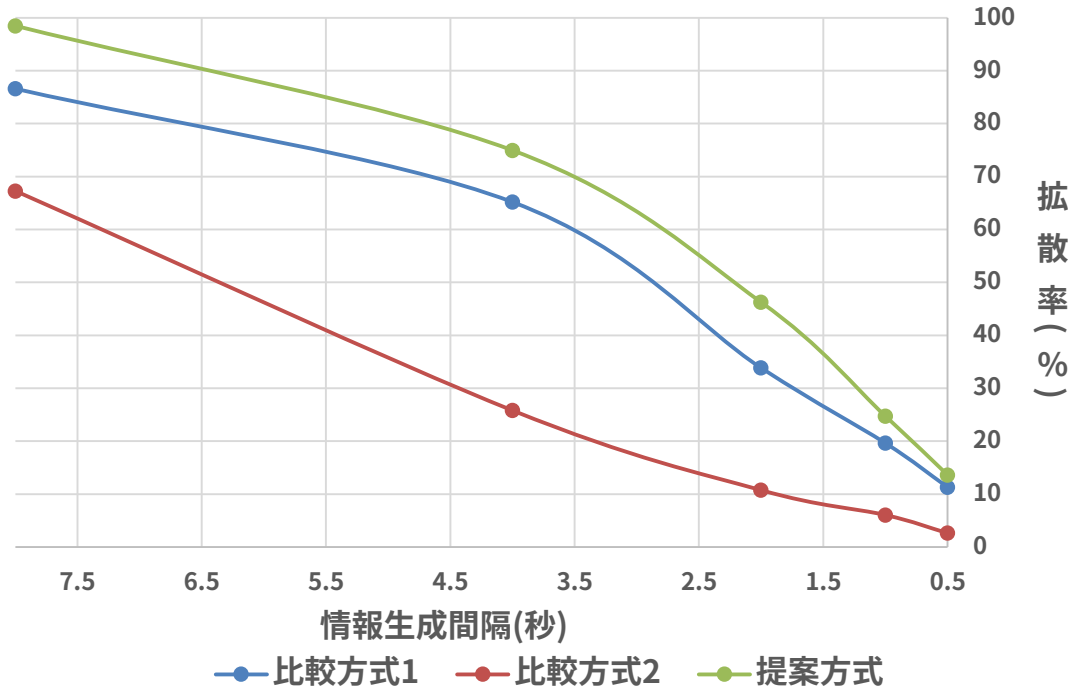

図 33 データサイズ 512byte における拡散率

6.5. 提案方式の機能別評価と考察

6.5.1. PRIORITY_MAX の最適値評価と考察

図 34 に PRIORITY_MAX を変化させた場合の情報生成間隔と拡散率の変化を示す.この結果から PRIORITY_MAX を高く設定した時，通信範囲内にいる多くの車両に中継優先度を割り当てるため，仮に，中継に最適な車両が情報を転送できなかった場合でも中継優先度の低い車両が転送をかけることで情報の拡散が止まってしまうことを防げる利点があるため，情報生成頻度が長い場合では拡散率を高く保てることがわかった.

一方で，PRIORITY_MAX を低く設定した時，中継に最適な車両のみが中継可能な車両となることで多くの通信が衝突することになる情報生成頻度が短い場合では，冗長な通信が減りかつ 1 回の通信で多くの車両へ情報を拡散できることで拡散率が高くなっていると考えられる.

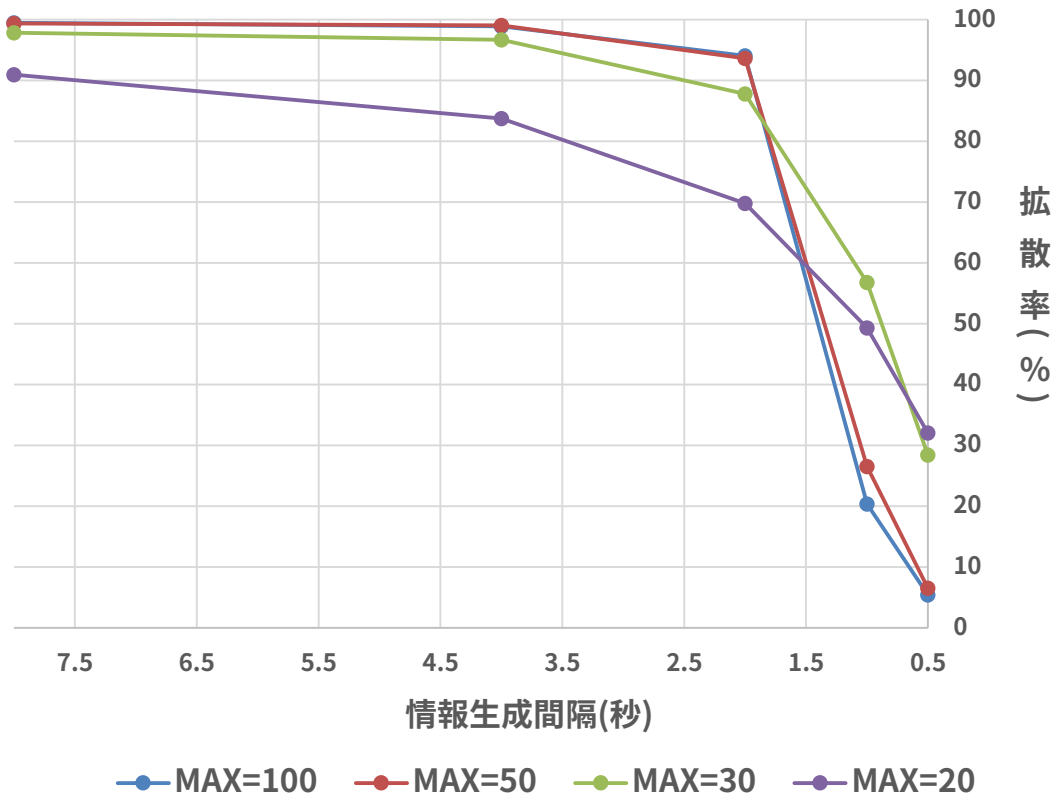

図 34　PRIORITY_MAX の変化と拡散率

図 35，図 36 に PRIORITY_MAX を変化させたときの総送信回数とエラーレートを示す．全ての情報生成間隔で PRIORITY_MAX=20 の時に送信回数の削減率とエラーレートの低減率が最大となり，PRIORITY_MAX=100 の場合と比較して情報生成間隔 1s の時が最大で送信回数では 68.3%，エラーレートでは 51.3%低減している．

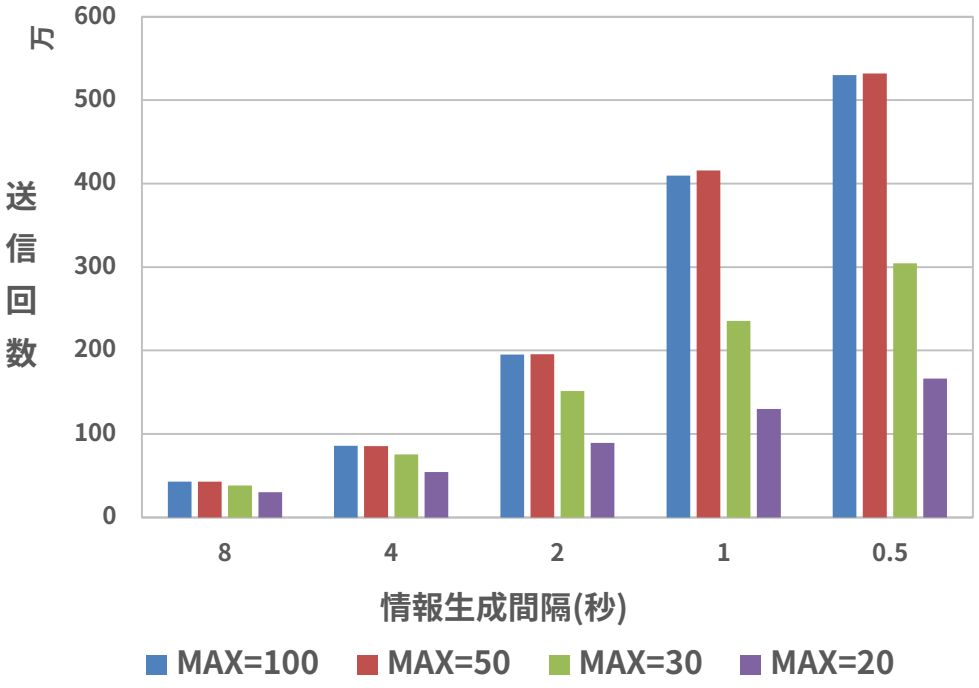

図 35　PRIORITY_MAX の変化と総送信回数

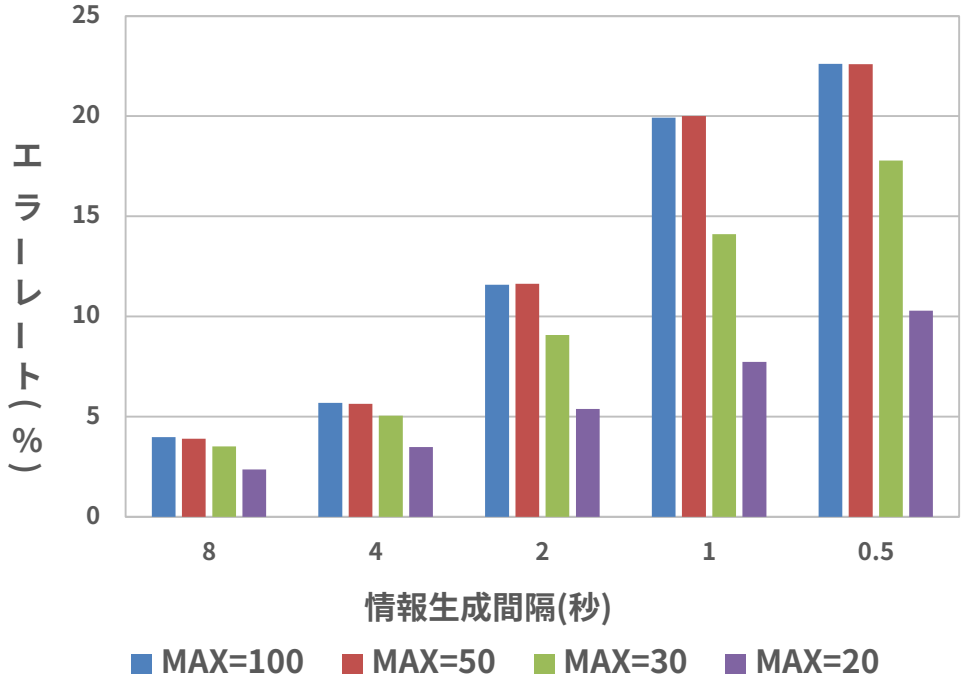

図 36　PRIORITY_MAX の変化とエラーレート

6.5.2. 交差点車両への特別優先度付与機能評価と考察

図 37 に交差点に存在する車両に特別な優先度を割り当てる機能の有無と拡散率の変化を示す.交差点車両への優先度割当により情報生成間隔 0.5s の時に最大で 447.7%拡散率が向上している.

情報生成間隔が長い場合，PRIORITY_MAX は高く設定される．交差点車両への特別優先度の有無にかかわらず中継車両として選択されることで拡散率に大きな差はでない.一方で，情報生成間隔が短くなると動的に PRIORITY_MAX が変化し低い値に設定されるため交差点内の車両は中継可能車両として選択されない場合がある.そのため，交差点車両に中継優先度を付与することで拡散率の向上が実現されていると考えられる.

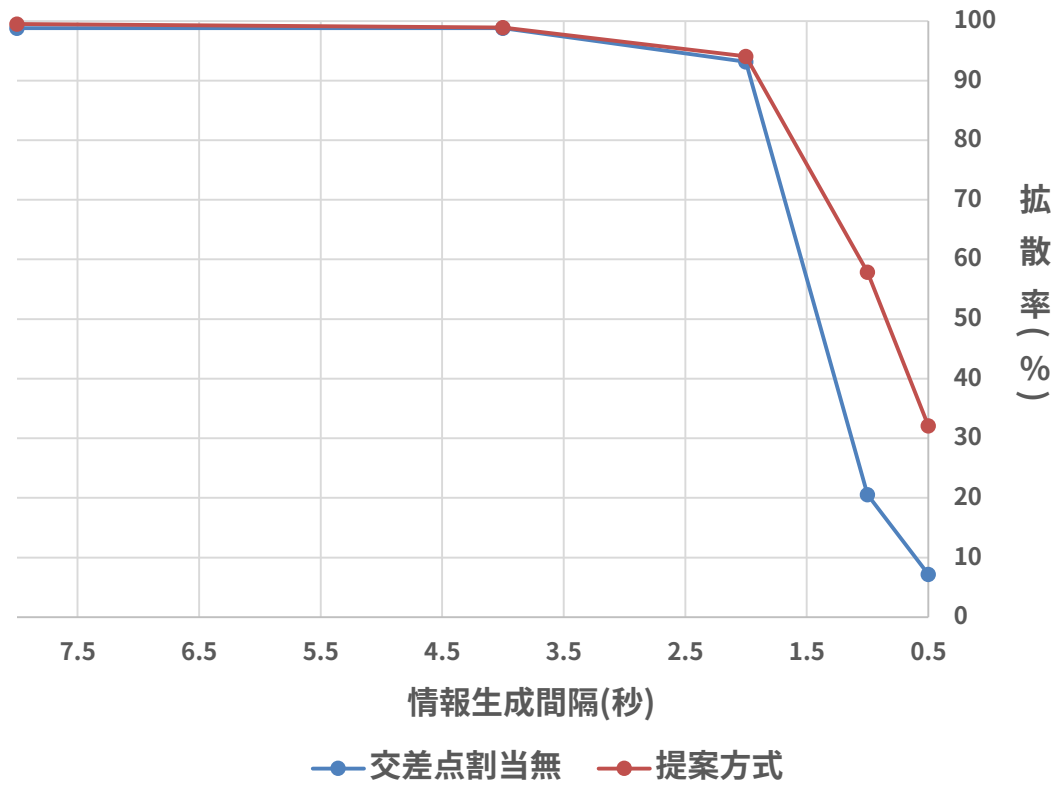

図 37 交差点中継割当の有無と拡散率

図 38 図 39 に交差点優先度割当の有無と総送信回数およびエラーレートを示す.情報性生間隔 8s-2s では拡散率と同様に送信回数，エラーレートともに同等の結果となっている.一方で，情報生成間隔 1s では交差点優先度割当により総送信回数は 43.9%削減できており，また，エラーレートでも 36.6%低減できている.

図 37 の拡散率評価と図 38 の送信回数評価から，提案方式では情報生成間隔 8s-2s の場

合でも 6-8%の送信回数を削減できている．情報生成間隔 8s-2s のような通信トラフィックに空きがある状態では交差点優先度を割り当てず交差点車両が中継を行わない場合でも回り込みによって情報が伝えられている可能性が考えられる．

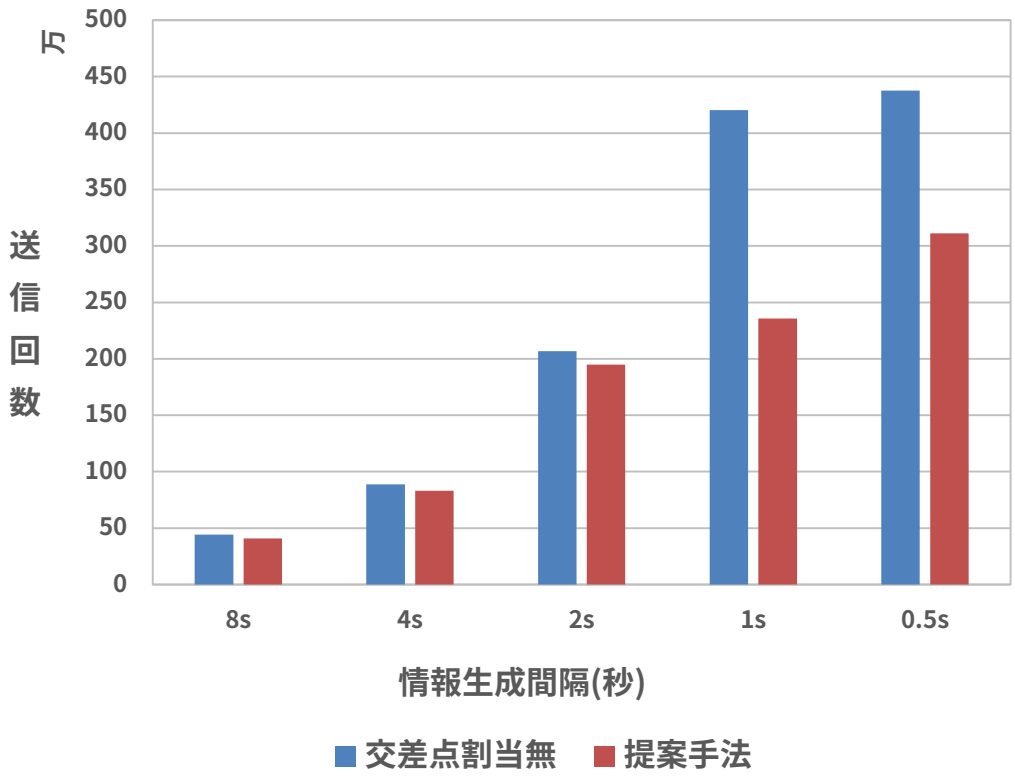

図 38 交差点中継割当の有無と総送信回数

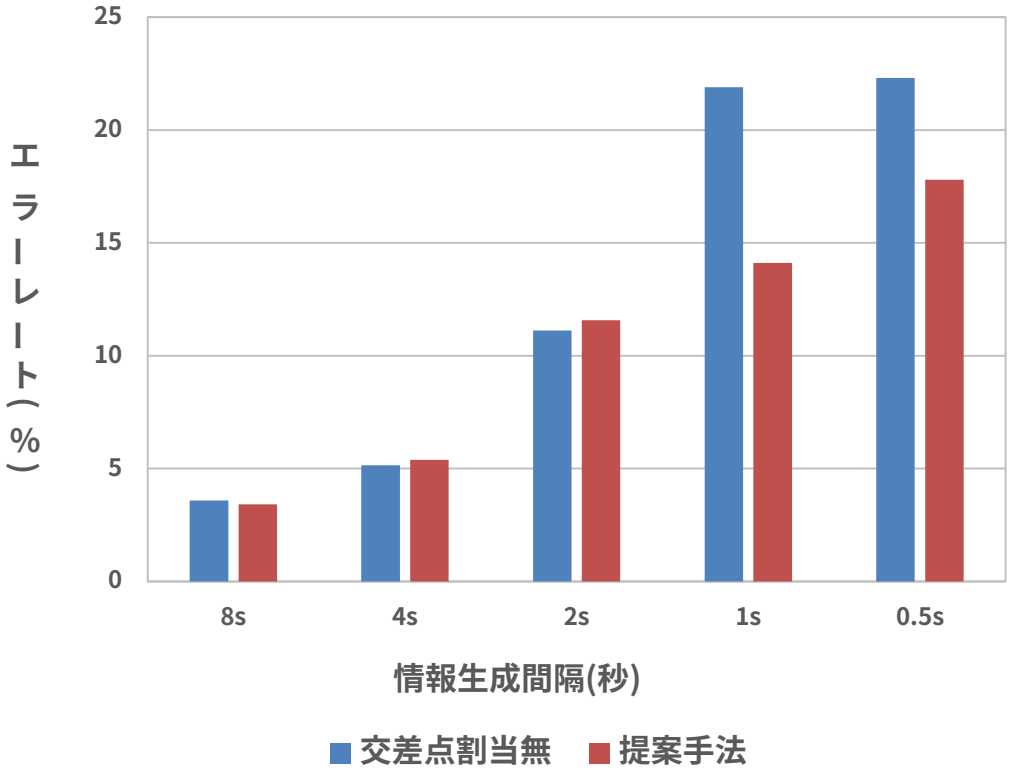

図 39 交差点中継割当の有無とエラーレート

6.6. 各車両が取得する位置情報の正確性

700MHz の位置情報の交換においてどの程度の正確性が保たれるかを確かめるために簡易なシミュレーション評価を行った.主なシミュレーションの設定値を表 9 に示す.

表 9　700MHz 帯シミュレーション設定値

項目	説明	値
通信方式	通信に利用するプロトコル	IEEE 802.11p
伝送速度	通信の速度	3Mbps
周波数帯	通信に使用する周波数帯域	760MHz
パケットサイズ	拡散させる情報の大きさ	128byte
最大ホップ数	何回まで中継を許すか	1
シミュレーション時間	シミュレーションの時間	120秒
試行回数	シミュレーションの反復回数	10回

700MHz 帯を利用して 0.1s 間隔に位置情報を交換した場合のエラーレートを図 40 に示す.図 40 よりエラーレートは 2%-2.3%で安定して推移していることがわかる.この結果から累積値を考えると 0.2s 間隔で 99.94% - 99.96%の車両が正確な位置を把握できていると推測できるため，提案方式のシミュレーションに与える影響は少ないと考えられる.

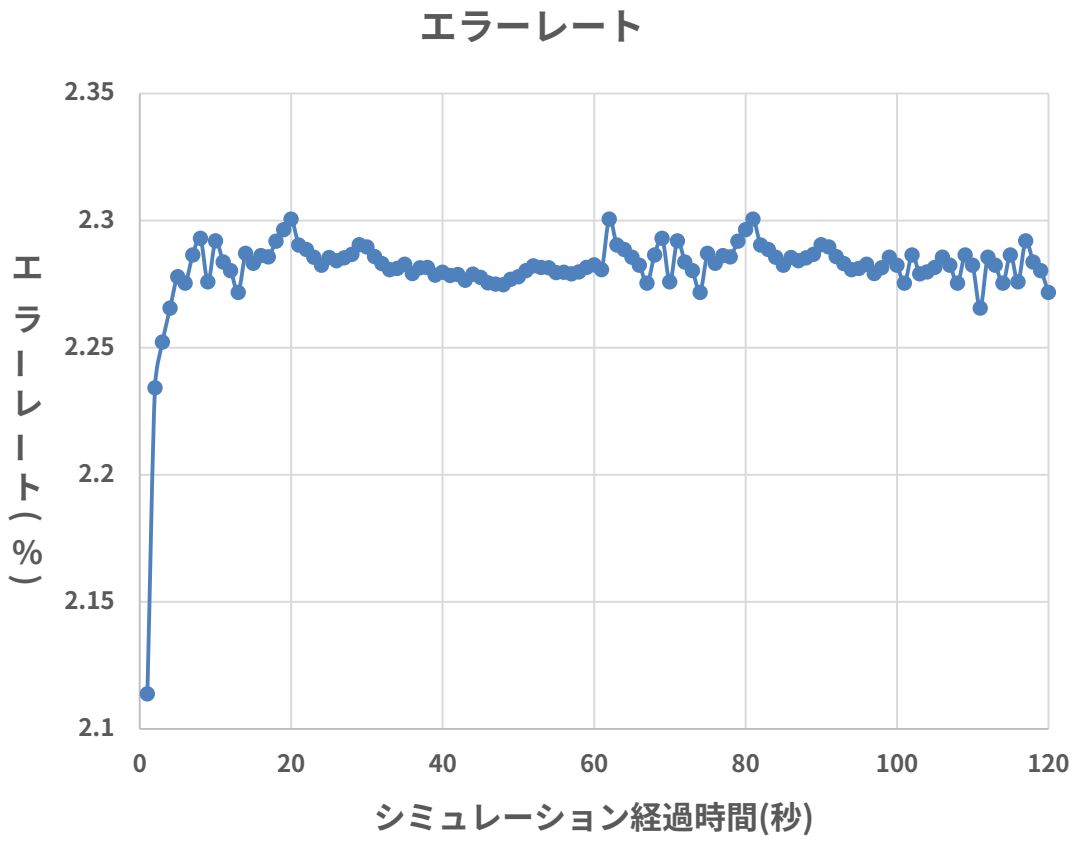

図 40　700MHz 通信における経過時間とエラーレート

7. 結論と今後の課題

7.1. 結論

本論文では，位置情報を用いた順序付けによる中継車両の選択方式について提案を行った.提案方式により以下の点を改善することをシミュレーションにより検証した.

- 比較方式 1(Counter-Based Scheme)よりも拡散率が最大で 35%向上した.
- 比較方式 1 よりも通信回数を最大で 20.5%削減した.
- 比較方式 1 よりも遅延時間を最大で 55.3%減少させた.
- 比較方式 2(Basic Distance-Based Broadcast)より拡散率が最大で 1065%向上した
- 比較方式 2 より通信回数を最大で 28.25%削減した.
- 比較方式 2 より遅延時間を最大で 62.9%減少させた.

また提案方式での各機能の評価によって PRIORITY_MAX の動的変更や交差点車両への優先度付与について考察した結果，以下のことがわかった.

- PRIORITY_MAX の動的変更は，通信トラフィックが混雑していない場合には情報の拡散が止まることを防ぎ，通信トラフィックが混雑している場合には最適な中継車両のみが情報を送信することで拡散率の向上に寄与している.
- 交差点車両への優先度付与は，通信トラフィックが混雑している場合に PRIORITY_MAX が低下した際でもより多くの車両へ通信のできる交差点車両が中継できることで拡散率の向上に寄与している.

以上の結果から，提案方式では既存方式と比較して，拡散率向上と平均遅延時間低減，情報送信回数の削減という観点から効率的な情報拡散が可能になったと言える.

7.2. 今後の課題

7.2.1. 各車両の位置情報の齟齬による影響

700MHz の電波伝搬に関してはいくつかの先行研究[20][21]があり，5.8GHz と比較してより広範囲での情報伝達が可能であることがわかっている.そのため，本論文のシミュレー

ションでは，各車両が正確な位置情報を取得している前提で行った.また 6 章でも 700MHz 帯を用いたシミュレーションにより 0.2s 間隔で 99.94% - 99.96%の車両が正確な位置を把握できという結果からも提案方式において位置情報の齟齬による大きな影響はないと考えている.しかし，700MHz 帯と 5.8GHz 帯の通信を行う車載器を統合した場合，チャネルの切り替え等に関わる処理にオーバヘッドが発生し，位置情報の取得の遅延が考えられるためそのような点も考慮し，シミュレーションを行う必要がある.

7.2.2. 緊急度を持った情報の優先転送制御

本提案では，拡散すべき情報はすべて均一なものとして扱った.しかし，局所的交通情報共有方式では，様々な交通情報を扱うため，その緊急度が異なる場合が考えられる.その際でも，緊急度の高い情報は，より多くの車両に迅速に届けられるべきでありそのような緊急度を考慮した優先転送制御方式についても検討および適用する必要がある.

7.2.3. 実環境に近いシミュレーションによる検証

本論文でのシミュレーションは，既存の論文による実験よりも，信号機の設置により都市部の交通流に近い環境を模して行った.今後は，車線数や交差点間隔の異なる道路など実際に存在する都市の交通を模したシミュレーションにより提案方式の有効性を確認する必要がある.

謝辞

本研究を遂行するにあたり，大変多くの方からのご協力，ご支援いただき，ここに深く感謝の意を表します.

本研究を行うにあたり，博士課程前期の２年間を通じた研究ならびに修士論文執筆の親身な指導を賜りました電気通信大学情報理工学研究科 指導教員の小花貞夫 教授に深く感謝致します.また副指導教員の沼尾雅之 教授には修士論文執筆における助言を賜りましたことに感謝いたします.湯素華 助教には研究への的確な助言を賜りましたことに感謝いたします.

私の学生生活を支えて修士として研究に没頭することに尽力していただいた両親および兄弟にも感謝いたします.最後に，研究室での生活で苦楽を共にし，充実した時間を共有することができた小花・湯研究室の学生の皆々様に深く感謝いたします.

参考文献

[1] 宮本進生, 四方博之, ｼｬｸﾞﾀﾞﾙｵﾕｰﾝﾁﾒｸﾞ, ﾇﾘ ｼﾗｼﾞ ﾏﾊﾀﾞｯﾄﾞ, 大山 卓, 三浦 龍, 小花貞夫, CDMA vs. TDMA: 安全運転支援のための車車間通信システム特性評価, 電子情報通信学会論文誌, Vol.J93-A, No.7,pp.474-484 ,July 2010

[2] GokhanKorkmaz, EylemEkici , Fusun O zguner, Umit O zguner: “Urban Multi-Hop Broadcast Protocol for Inter-Vehicle Communication Systems,” Proc. of of the 1st ACM international workshop on Vehicular ad hoc networks VANET '04 , pp.76-85 , 2004

[3] Chakkaphong Suthaputchakun, Aura Ganz, ”Priority Based Inter-Vehicle Communication in Vehicular Ad-Hoc Networks using IEEE 802.11e,” Proc. of the IEEE Vehicular Technology Conference, 2007 ,pp.2595-2599.

[4] Chakkaphong Suthaputchakun, Zhili Sun, “Priority based Routing Protocol with Reliability Enhancement in Vehicular Ad hoc Network,” Proc.of the 2nd International Conference on Communications and Information Technology, pp.186-190 , 2012.

[5] M. Torrent-Moreno, D. Jiang, and H. Hartenstein, “Broadcast Reception Rates and Effects of Priority Access in 802.11-Based Vehicular Ad-Hoc Networks,” Proc. of the 1st ACM International Workshop on Vehicular Ad hoc Networks, ｐpp10-18 , Oct. 2004.

[6] 姜巍, 川瀬悠, 若山公威, 岩田彰, 白石善明, 車車間通信におけるコリジョン数を削減するフラッディング方式の提案と評価, 情報処理学会研究報告, 2008(25), pp.51-57 , 2008-03-07

[7] 吉川潤, 齋藤淑,小花貞夫,車車間通信を利用した局地的交通情報の共有方式に関する一考察, マルチメディア,分散協調とモバイルシンポジウム 2013 論文集,pp.1503-1509 , 2013.

[8] 瀧本栄二, 近藤良久, 板谷聡子, 鈴木龍太郎, 小花貞夫, マルチチャネルフラッディングの実装と評価, 電子情報通信学会技術研究報告, 情報ネットワーク 107(148), pp.91-94, 2007-07-12

[9] Yacine Khaled, Ines Ben Jemaa, Manabu Tsukada ,Thierry Ernst” Application of IPv6 multicast to VANET,” Proc. of the 9th International Conference on Intelligent Transport Systems Telecommunications, 2009

[10] Christopher Ho , Katia Obraczka , Gane Tsudik , KumerViswanath,”Flooding for Reliable Multicast in Multi-Hop AdHoc Networks,” Proc. of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications DIALM '99, 1999

[11] Wen-Hsing Kuo , Shiqi Dong ,Jen-Shian Huang ,”Pheromone-Based V2V Unicast Routing Scheme in VANETs,” Proc. of the International Conference on Connected Vehicles and Expo (ICCVE), 2013

[12] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu ,“The Broadcast Storm Problem in a Mobile Ad Hoc Network,” Proc. of the 5th Annual ACM/IEEE International Conference on Mobile computing and networking MobiCom '99, pp.151-162 , 1999.

[13] Brad Williams, Tracy Camp”Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks”, MobiHoc '02 Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, pp.194-205 , 2002

[14] Sang-woo Chang, Sang-sun Lee,” A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication”, Proc. of International

Conference on Convergence and Security (ICITCS), 2013

[15] The Qualnet Network Simulator, http://www.scalable–networks.com.
[16] Alam. M, and Sher. M. Husain. S.A, "Integrated Mobility Model (IMM) for VANETs simulation and its impact," Proc. of the International Conference on Emerging Technologies, pp.452-456, October 2009.
[17] M.Chegin, and M.Fathy, "Optimized routing based on mobility prediction in wireless mobile adhoc networks for urban area," Proc. of the fifth International conference on Information Technology: New Generations (ITNG), pp 390-395, April 2008.
[18] SUMO, Simulation of Urban MObility, http://sourceforge.net/apps/mediawiki/simo
[19] Space-Time Engineering,Scenargie Simulator, https://www.spacetime-eng.com/en/
[20] Hisato Iwai,Satoshi Goto ,"Multipath Delay Profile Models for ITS in 700MHz Band," Proc. of the IEEE Vehicular Technology Conference VTC Fall 2011 ,Sept.2011
[21] Ryoji YOSHIDA, Hisato IWAI, and Hideichi SASAOKA ," Evaluation of Propagation Loss Difference between 5.8GHz and 700MHz Bands in V2V Communication Environments," Proc. of the Asia-Pacific Microwave Conference (APMC), Nov. 2013.

付録A.発表原稿

本研究は情報処理学会 第 60 回 高度交通システム研究会にて発表した．その原稿を付録として添付する．

狭域交通情報共有のための車車間通信における車両位置情報に基づく効率的な中継転送方式の提案

吉川潤† 湯 素華† 小花 貞夫†

ITS(高度交通システム)では，車車間通信により，お互いの位置・速度情報等を頻繁に交換して衝突を防止するシステムの研究開発が行われてきているが，今後は，安全運転のみならず，エコドライブや運転の快適性･利便性向上を図るシステムの実現も強く期待される．本稿では，ドライバが道路や交通の状況に応じて走行できるように，車車間通信により渋滞や事故等の局地的な交通情報を周辺車両で効率的に共有させることを目的として，中継車両選択における従来方式の問題点であったほぼ同位置にある車両間での通信の衝突を防止するために，700MHz 帯を使って共有される位置情報から順序付けによる中継転送方式を提案する．シミュレーションにより提案方式と既存方式を比較し，35%の拡散率向上，55%の遅延時間の低減を実現した．

Relative-position-based collision-free relay selection for efficient local sharing of traffic information

JUN YOSHIKAWA† SUHUA TANG† SADAO OBANA†

In ITS (Intelligent Transport Systems), inter-vehicle communications (IVCs) are used to exchange position and speed information between vehicles to avoid collisions. Besides this support system for safe driving, new functions like eco-driving, and comfort and convenience of driving are also strongly expected. This paper aims at efficient, local diffusion of congestion and accident information so that drivers can learn road and traffic conditions before hand, and change their routes accordingly. IVCs take place over two bands: 700MHz for safe driving and 5.9GHz for comfort driving. With the 700MHz band, vehicles learn the same local map of vehicles. This information provides the relative position between vehicles, and is used for relay selection for the communications in the 5.9GHz band. More specifically, potential relay vehicles are sorted according to their distances from the sending node, and the farthest, available vehicle is selected as the relay. The proposed scheme is implemented via network simulator. Extensive evaluations confirm that the proposed scheme achieves both higher diffusion rate and lower latency, compared with state-of-the-art methods.

1. はじめに

ITS(高度道路交通システム)では，車両間通信により，お互いの位置・速度情報等を頻繁に交換して衝突を防止するシステムの研究開発が行われてきている[1]が，今後は，安全運転のみならず,エコドライブや運転の快適性･利便性向上を図るシステムの実現も強く期待される． 本稿では，ドライバが道路や交通の状況に応じて円滑かつ快適に走行できるように，車車間通信により渋滞や事故等の局地的な交通情報を周辺車両で効率的に共有させるための車両における中継転送方式の提案し，シミュレーション評価により有効性を確認する．

2. 交通情報共有の概要とその課題

狭域交通情報共有の概要を図 1 に示す．狭域交通情報共有とは，車車間通信や路車間通信を用いて車両が走行中に捕捉した様々な交通的な事象に関する情報を共有することで円滑で快適な運転を実現する仕組みである．共有する情報は事故・渋滞情報や緊急車両情報，路面情報，急制動情報，ドライバリクエストなどを想定し，情報の種類や位置データなどの小容量データを中心とする．またマルチホップ中継を行うことで遠くの車両に情報を伝達する．

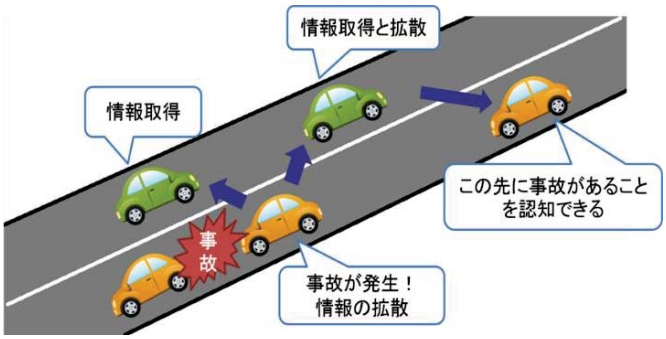

図 1 交通情報共有方式の概要

車車間通信を利用して交通情報の共有を目的とする研究は数多く存在[2-8]するが，車車間通信において車両の急制動や安全運転支援や渋滞情報などの運転快適性を向上させるための通信では，通信の対象は特定の車両ではなく，情報を拡散させる範囲内に存在する不特定多数の車両へ伝達する必要がある．さらに各車両が高速に移動していることを考えるとモビリティに利点のあるブロードキャストによる情報の拡散が望ましいと考えられる．ブロードキャストによる情報の拡散を実現する上での問題点として遠方まで情報を伝達させる上でどの車両に中継させるかという課

†電気通信大学 大学院情報理工学研究科
The University of Electro-Communications - Graduate School of Informatics and Engineering

題がある．

中継車両の選択方式として，既に送信者と受信者の距離情報を用いてより遠方の車両を中継車両として選択する方式[2-6]が提案されているが，都市部での高密度な車両環境で複数車線存在する場合などほぼ同位置に車両がいる場合，それぞれが中継するパケットが衝突し，配信効率を低下させるという問題がある．この問題を解決するために本稿では距離ではなく順番付けによる中継転送方式を提案し，効率的で低遅延での狭域交通情報の拡散を実現する．

3. 先行研究と問題点

3.1 既存の中継ノード選択方式

移動体アドホックネットワークでの既存の中継ノード(車両)の選択方式について以下にまとめる．

(a)Flooding

パケットを受け取ったノードはそのパケットが既に受け取ったパケットかどうかに関わらず周囲のノードに中継を行う．

(b)Probabilistic Scheme[7]

確率的に中継を行うかを決定する方式である．ブロードキャストによるパケットを最初に受信したときに確率 P によって中継を行うかを判定する．P=1 の場合，Flooding と同様の振る舞いになる．

(c)Counter-Based Scheme[7]

中継を行う場合にランダムな待ち時間を設定し，パケットの送信を開始する前に他のノードから同じパケットを n 回受信することで送信をキャンセルする方式である．n は閾値によって決められる．

(d)Distance-Based Broadcast[8]

VANET における距離ベースのマルチホップブロードキャストの中継車両選択方式であり，パケットを受信してから中継するまでに受信者と送信者の距離に応じて変化する線形的な待機時間を用いることで実現する．受信者と送信者の距離が長ければ送信待機時間は短くなり，優先的に送信できる．もし，送信待機中に同一の情報を受信した場合は他の車両が中継したことを認知し，送信をキャンセルして冗長な通信を抑制する．

3.2 先行研究の問題点

1) Probablistic Scheme，Counter-Based Scheme の問題点

ランダムな待ち時間の設定により，中継すべきノード(車両)が単一に絞られないことにより冗長な通信が行われる点や送信者から最も遠いノード(車両)が選択されない可能性があることから遠方のノード(車両)へ情報を伝達する際に中継回数が増え，効率的でない中継が行われる問題点がある．

2) Distance-Based Broadcast の問題点

ほぼ同位置に存在する車両が複数ある場合，それらの車両による中継パケットが衝突してパケット到達率が低下する問題点がある．

4. 位置情報に基づく順序付けによる中継転送方式

4.1 提案方式の概要

前章の問題点を解決する方式として，安全運転支援システムでの 700MHz 帯によって交換される周囲の車両の位置，速度，移動方向の情報に基づいた順序付けによる中継転送方式を提案する．また，本稿で扱う交通情報は，ITS 利用が許可されている 5.8GHz 帯にて通信するものとする．

4.2 機能詳細

4.2.1 車両相互の車両位置把握

700MHz 帯を利用して，100ms 毎に周囲の車両と位置，速度，移動方向の情報および車両 ID を交換する．それを元に各車両は周囲の車両の位置を格納する．車両位置を相互に交換するパケットには，ARIB より定められる STD T109[9]に準拠する以下の表 1 の情報が含まれるものとする．

表 1 位置情報パケットの構造

項目	説明
NodeId	車両番号
NodePositionX	車両の X 座標
NodePositionY	車両の Y 座標
Speed	車両の速度
DirectionofMove	車両の進行方向
CurrentTime	現在時刻

4.2.2 中継車両の選択(基本的な選択方法)

各車両は前述の相互の車両位置把握機能で格納した位置情報を使用し，送信車両の位置と自車両の位置から中継優先度を決定する．中継が必要な情報を受け取った車両は受信したパケットから送信者の位置情報を取得し，自車両との距離を計算する．その後，周囲の車両と送信車両の距離を算出し，自車両の中継優先度を決定する．パケットを受信したすべての車両が同一位置情報に基づいて処理をすることにより周辺車両間で共通の中継優先度が共有される．中継優先度が 1 の時，最も優先され，優先度の最大値は後述の PRIORITY_MAX として動的に変更する．図 2 に示す，距離により中継優先度を算出する既存方式とは異なり，図 3 で示すように送信元車両からの距離ではなく順番により決定する．算出した中継優先度に応じて，各車両の送信待機時間(後述)を設定することで中継機会に差をつけることにより既存方式では通信が衝突していた問題を回避可能と

する．中継優先度は送信者から最も遠方にある車両が最も高く，等距離に他の車両がいる場合は車両 ID を比較し，小さい車両を優先する．また，各車両は，中継前に他の車両の中継を確認したら中継をキャンセルすることで冗長パケットの伝搬を抑制する．

また，位置情報は 700Mhz 帯で交換されるため，狭域交通情報を共有する 5.8GHz 帯よりも遠くまで電波が到達することから，各車両が取得している位置情報に基づきすべてに中継優先度を付与した場合，実際の交通情報の通信では電波が到達しない可能性がある．そのため，想定される 5.8GHz での通信可能距離内(例えば 380m と設定)でのみ中継優先度を付与することで適切な中継車両の選択制御を可能とする．

（1） 送信待機時間

中継優先度により算出される送信待機時間は MAC（Media Access Control）層に実装される．MAC 層での送信待機時間はバックオフ時間と呼ばれ，Counter-Based Scheme では，式(1)で表されるように DIFS(Distributed coordination function Interframe Space)と呼ばれる固定長の待機時間とコンテンションウィンドウの範囲内で選択されるランダム値×スロットタイムにより決定されるが，提案方式では式(2)により決定する．

$$\text{WaitTime(待機時間)} = \text{DIFS} + \text{Random\_NUM}(\text{CW\_MIN}, \text{CW\_MAX}) * \text{SLOT\_TIME} \quad (1)$$

CW_MIN:コンテンションウィンドウ最小値
CW_MAX:コンテンションウィンドウ最大値

$$\text{WaitTime(待機時間)} = \text{DIFS} + \text{RelayPriority} * \text{SLOT\_TIME} \quad (2)$$

RelayPriority:中継優先度

（2） 交差道路に存在する車両に対する中継優先度割当

交差道路に存在する車両の中継は車両の進行方向とは異なる方向への情報拡散に有効であるため，情報を受信した車両が送信者から異なる方向に存在する場合，直線方向の車両へ割当てられる中継優先度とは重複しない中継優先度を割当てる．

（3） 中継優先度の割当車両台数の動的変更

中継優先度を設定し，中継が可能となる車両の台数を PRIORITY_MAX として表現する．中継優先度は，情報を受信した全ての車両に付与するわけではなく，通信状況によって変化する PRIORITY_MAX の値の台数に付与する．割当台数(PRIORITY_MAX)が 3 と 5 の時の割当例を図 4 に示す．

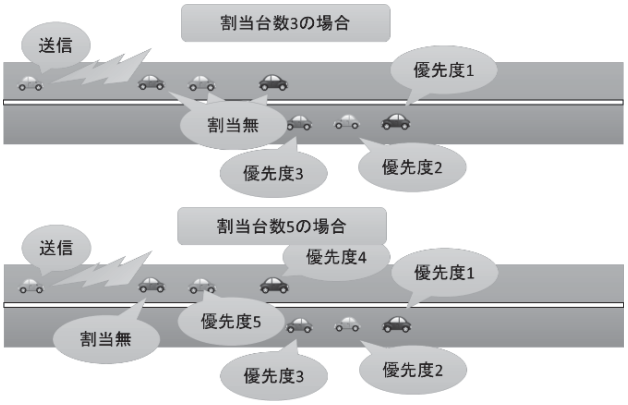

図 4 中継優先度の割当例

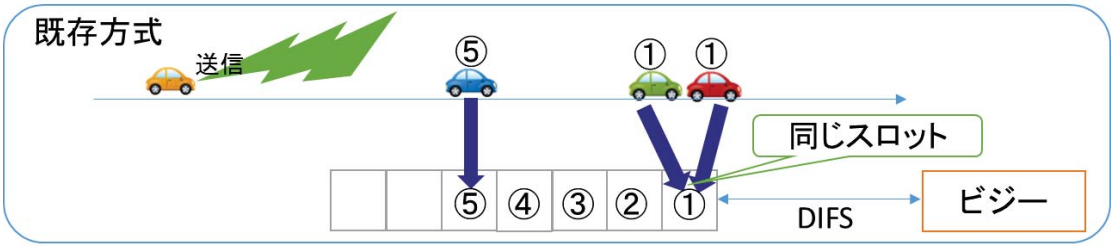

図 2 Distance Based Broadcast による中継車両選択

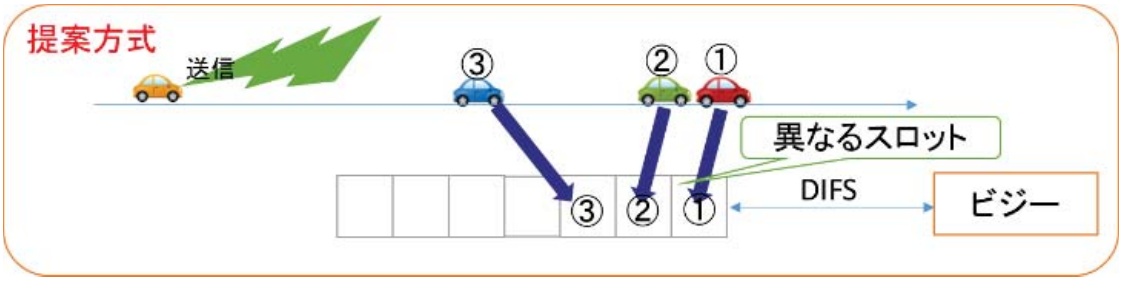

図 3 提案方式による中継車両選択

割当台数を小さくした場合，送信者から遠い車両のみが中継することで効率的な情報の拡散が可能であるが，その車両が隠れ端末問題等により情報を受信できなかった場合，それ以上情報の拡散が行われないという問題点がある．

割当台数を大きくした場合，中継が行われたことを通信の衝突等によって確認できなかった場合，不要な中継を行ってしまう問題がある．

提案方式における処理フローを図 5 示す．パケットは表 2 に示す情報を含むものとする．

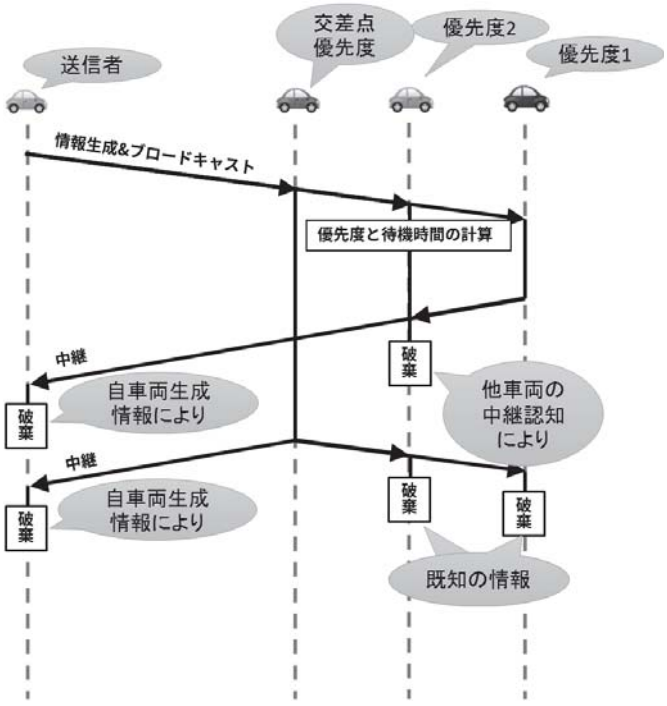

図 5 提案方式の処理フロー

表 2 パケットの構造

項目	説明
BroadcastTime	情報を生成した時間
SourceNodeId	情報生成した車両番号
SourceNodePositionX	情報を生成した車両の X 座標
SourceNodePositionY	情報を生成した車両の Y 座標
maxHopCount	最大ホップ数
counterThreshold	中継回数の閾値
RelayPriority	中継優先度
RelayNodeId	中継車両の車両番号
EventData	イベントデータ

5. シミュレーション評価

5.1 シミュレーション概要

提案方式の有効性をシミュレーションにより検証するため，提案方式と 3 節で述べた比較方式を汎用ネットワークシミュレータ Scenargie[10]に実装した．比較方式は Counter-Based Scheme[7]， Distance-Based Broadcast[8]を対象とした．

提案方式と比較方式の評価項目は，情報生成間隔を変化させたときの，1)情報拡散率，2)総通信回数，3)エラーレート，4)平均遅延時間とする．また，提案方式における中継優先度の割当車両台数の動的変更機能と交差道路車両への中継優先度割当機能の評価，ならびに 700MHz 帯を利用した位置情報交換の正確性評価も行った．評価では，シミュレーション時間内における平均値を使用する．

5.1.1 比較方式の実装

（1）Counter-Based Scheme(比較方式 1)

各車両は情報を生成し，発信する．情報を受信した車両は，ランダムな送信待機時間の終了後に送信する．送信待機時間は以下の式による．

$$\text{WaitTime}=\text{Random}(0.1 – 0.5)*\text{SECOND} \quad (3)$$

送信待機時間に同一のパケットを閾値 n 回受信することで送信をキャンセルする．今回，n=1 とする．

（2）Distance-Based Broadcast(比較方式 2)

通信可能距離を任意の分割するセクタ数で除し，送信車両と受信車両の距離を計算し，自車両が何番目のセクタに存在するかにより待機時間を設定する．待機時間は以下の式による．なお，ここでは，SPLITNUM=100 ， Distance＝380 とする．

$$[\text{MySector}] = \frac{\text{Distance} * \text{MAXRANGE}}{\text{SPLITNUM}} \quad (4)$$

$$\text{WaitTime} = \frac{(0.5 - 0.1)}{SPLITNUM} * MySector \quad (5)$$

MySector:自分のセクタ番号 ， Distance:送信者からの距離，SPLITNUM:セクタ分割数 ， MAXRANGE:通信可能距離．

5.2 シミュレーション条件

交差点間隔 400m で 4×4 のグリッド状のマップに 500 台の車両を配置する．道路以外の部分には建物が存在し，各交差点には信号機が設置する．各ノードはランダムに決定された初期位置からランダムに移動し，一定間隔で情報を生成する．電波伝搬モデルは建物による電波の遮蔽を考慮する ITU-R P.1411 の標準に従う．主なシミュレーションの設定値を表 3 に示す．

表 3 シミュレーション設定値

項目	値
通信方式	IEEE 802.11p
伝送速度	3Mbps
周波数帯	5.8GHz
パケットサイズ	128byte – 512byte
最大ホップ数	100
電波伝搬モデル	ITU-R P.1411
情報生成間隔	0.5s/台-8s/台
評価エリア	1.6km x 1.6km
交差点間隔	400m
信号の有無	有(全ての交差点)
車線数	2
車両位置	ランダム
車両台数	500 台
車両長	5m
車両速度	30km/h-60km/h
シミュレーション時間	120s
試行回数	30

5.3 情報拡散率に関する評価と考察

8s-0.5s まで情報生成間隔を変化させたときの平均拡散率を図 6 に示す．図 6 より，全ての情報生成間隔で提案方式が最も拡散率が高く，特に情報生成間隔が 0.5s の時に比較方式 1(Counter-Based Scheme)よりも拡散率が最大 35.0%向上している．また比較方式 2 (Distance-Based Broadcast)では 2.7%から 32%に向上している．シミュレーションでは，信号により交差点付近に車両が多く存在するため，比較方式 2 おいて通信が衝突しやすいことが原因で，比較方式 2 が最も悪い結果になったと考えられる．

また, 情報生成間隔 8s の時の方式ごとの拡散率分布を図 7 に示す. 提案方式では拡散率 100%の情報の個数が最も多い結果となったことより，最遠車両の中継やパケットの衝突を回避することでより多くの車両へ情報を拡散できていることが確認できる．

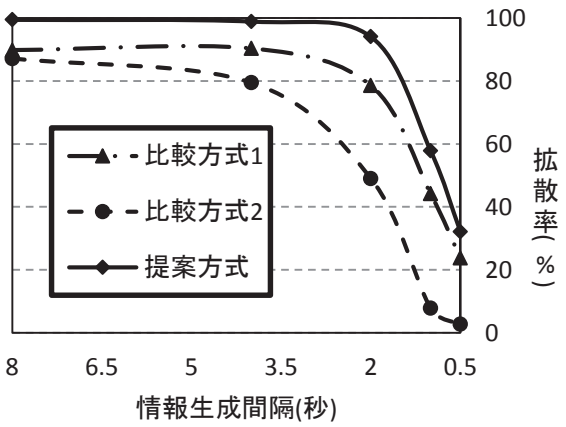

図 6　情報生成間隔と拡散率

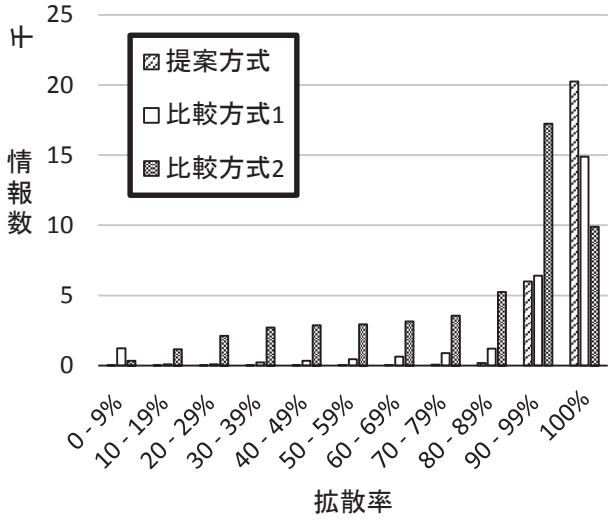

図 7 方式別の拡散率分布

5.4 通信回数と信頼性に関する評価と考察

図 8 に情報生成間隔を変化させたときの総送信回数を示した．8s-2s では提案方式と比較方式 1(Counter-Based Scheme)の総送信回数は同等であり，1s，0.5s 間隔では総送信回数を最大 20.5%削減できていることがわかる．全ての情報生成間隔で比較方式 2 の送信回数が多くなっており情報生成間隔が 1s の時に提案方式では最大 28.25%削減できている．

8s から 4s，4s から 2s の間隔で情報を生成する場合，時間当たりで倍の情報が生成されることより同等の拡散率を維持するためには, 総送信回数も約 2 倍になるはずである．しかし 2s から 1s および 1s から 0.5s では総送信回数の増加率は鈍化しており，通信トラヒックに空きがなくなっていることが予想される．

図 6 と図 8 を併せてみると比較方式 1 および比較方式 2 では提案方式よりも送信回数が多いにも関わらず拡散率が向上していない．この結果から，提案方式では一回の送信でより多くの車両が情報を取得できていることがわかる．対して，比較方式 1 および比較方式 2 では同一の情報が複数回中継されていることやパケットの衝突によって効率的な情報拡散が行われていないと予想される．

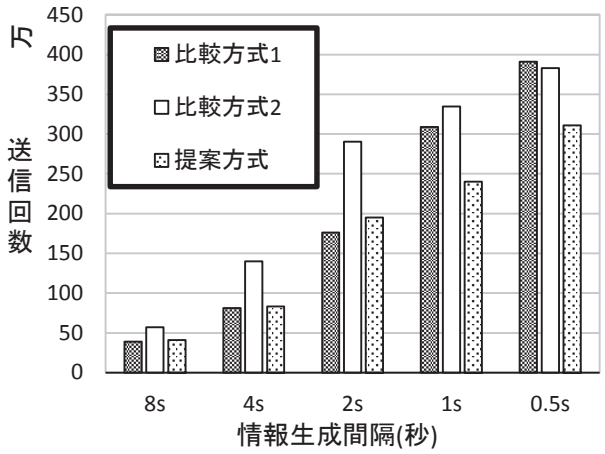

図 8 情報生成間隔と送信回数

図 9 より，情報生成間隔 1s，05s の時（高トラヒックの時)，提案方式ではエラーレートが比較方式 1(Counter-Based Scheme)および比較方式 2(Distance-Based Broadcast)よりも低い値で抑えられていることにより信頼性の高い通信ができていることがわかる．一方で，8s-2s の時にエラーレートが提案方式で若干高い値を示しているのは，トラヒックに空きがある場合には拡散率を高めるために中継優先度割当台数を高い値に動的に設定し情報の拡散に冗長性を持たせていることが原因と考えられる．

比較方式 2(Distance-Based Broadcast)については，信号待ちの車両など近くの位置に複数の車両が存在する場合，通信がかならず衝突してしまうためエラーレートでは提案方式および比較方式 1(Counter-Based Scheme)よりも高い値を示している．

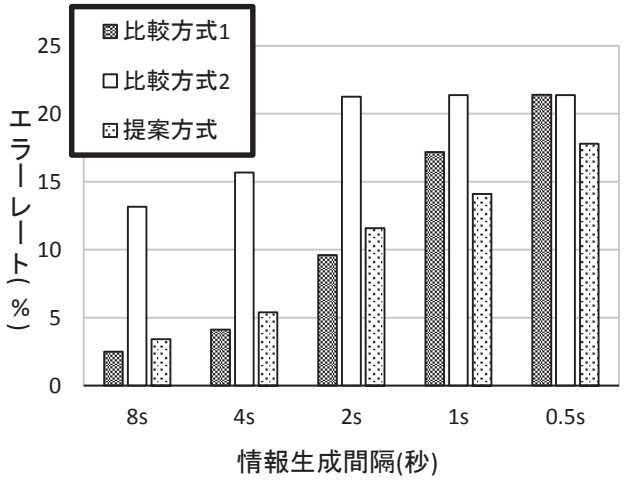

図 9 情報生成間隔とエラーレート

5.5 遅延時間に関する評価と考察

図 10 は情報生成間隔を変化させたときの平均遅延時間を示す．提案方式では比較方式 1(Counter-Based Scheme)よりも情報生成間隔 2s のときに最大で 55.3%，比較方式 2(Distance-Based Broadcast)よりも 62.9%遅延時間を低減している．提案方式では，送信者から最も遠くの車両を中継車両に選択することにより情報のホップ数を減少させる点や交差道路の車両を中継車両と選択することで遅延時間の増大を防げている点が大きく寄与している．また提案方式と比較方式 1 において情報生成間隔が 1s の時よりも 0.5s の時に遅延時間が減少している理由として，0.5s の場合にはトラヒックが圧迫され中継の際にデータが失われてしまうことで，少ないホップ数の情報の遅延時間のみが統計値として得られていることが原因と考えられる．比較方式 2，提案方式の情報生成間隔 2s から 1s の結果でも同様の現象が起きていると考えられる．

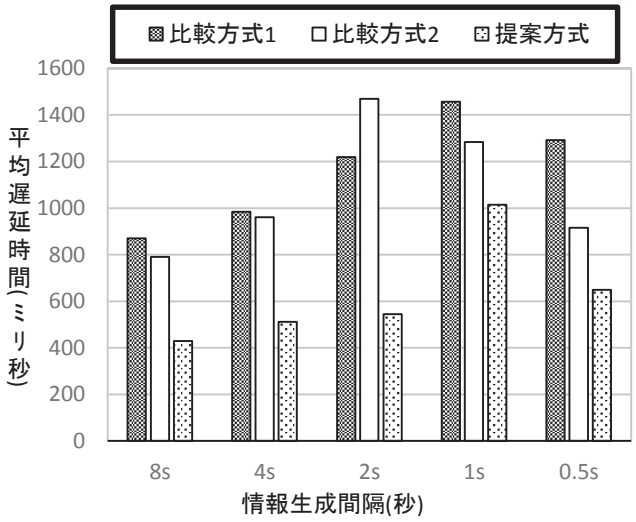

図 10 情報生成間隔と遅延時間

5.6 中継優先度の割当車両台数の動的変更に関する評価と考察

中継優先度の割当台数の最大値を 20， 50， 100 と静的な値に設定し，情報生成間隔を変化させたときの結果を図 11 に示す．結果から，割当台数を多く設定した場合，通信範囲内にいる多くの車両に中継優先度を割り当てるため，仮に，中継に最適な車両が情報を中継できなかった場合でもそれよりも中継優先度の低い車両が中継することで情報の拡散が止まることを防げる利点があり，情報生成間隔が長い場合では拡散率を高く保てることがわかる．

一方で，割当台数を少なく設定した場合，中継に最適な車両のみが中継可能な車両となることで，多くの通信が衝突することになる情報生成間隔が短い場合では，1 回の通信で多くの車両へ情報を拡散できることで拡散率が高くなっていると考えられる．

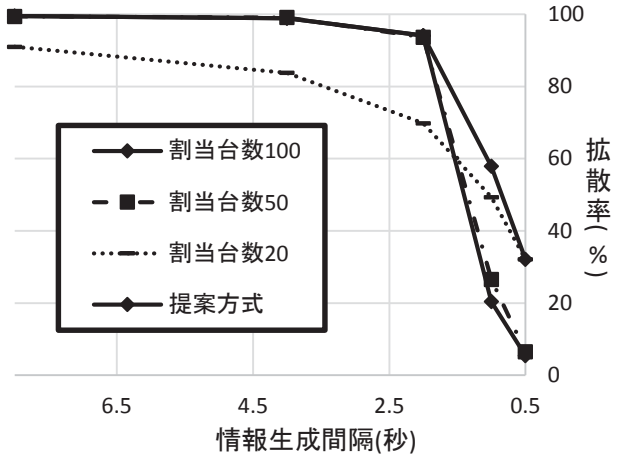

図 11 優先度割当台数の変化と拡散率

5.7 交差道路の車両への中継優先度の割当に関する評価と考察

図 12 には交差道路に存在する車両に中継優先度を割り当てる機能の有無と拡散率の変化を示す．情報生成間隔が長い場合，中継優先度の割当車両台数は多く設定されるので，交差道路車両への中継優先度割当機能の有無にかかわらず中継可能車両として選択される．そのため拡散率に大きな差はでない．一方で，情報生成間隔が短くなると中継優先度の割当車両台数は小さい値に設定されるため交差道路の車両は中継可能車両として選択されない場合がある．結果として，交差道路の車両に中継優先度を付与することで拡散率の向上が実現されていると考えられる．

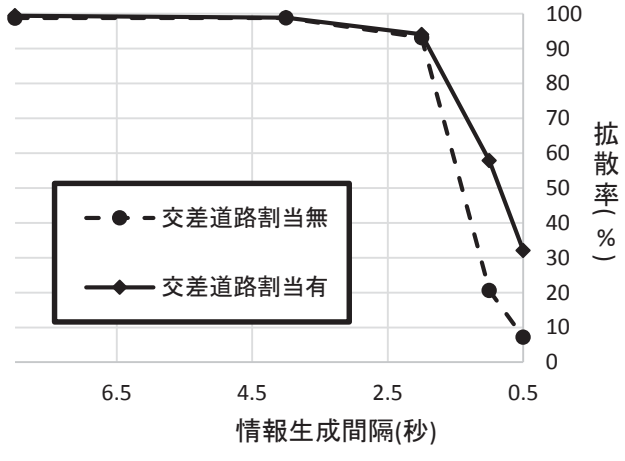

図 12 交差道路の車両への優先度割当の有無と拡散率

5.8 700MHz 帯を利用した位置情報交換の信頼性に関する評価

車両間で位置情報等と交換する 700MHz の電波伝搬では，5.8GHz と比較して広い範囲で通信可能であることがわかっている．そのため，シミュレーションでは，各車両が正確な位置情報を取得している前提で行った．しかし，実際に 700MHz 帯の位置情報の交換においてどの程度の正確性が保たれるかを確かめるために簡易なシミュレーションを行った．主なシミュレーションの設定値は表 4 の通りである．

表 4 700Mhz 帯シミュレーション設定値

項目	値
通信方式	IEEE802.11p
伝送速度	3Mbps
周波数帯	760MHz
パケットサイズ	128byte
最大ホップ数	1
シミュレーション時間	120s
試行回数	10 回

700MHz 帯を利用して 0.1s 間隔に位置情報を交換した場合のエラーレートを図 13 に示した．図 13 よりエラーレートは 2%-2.3%で安定して推移していることが読み取れる．この結果から累積値を考えると 0.2s 間隔で 99.94%-99.96%の車両が正確な位置を把握できていると推測できる，さらに提案方式で必要とされる位置情報の範囲は 5.8GHz の電波到達範囲でよいので更に高い信頼性で位置情報の交換が可能と思われるため，位置情報の齟齬が提案方式に与える影響は少ないと考えられる．

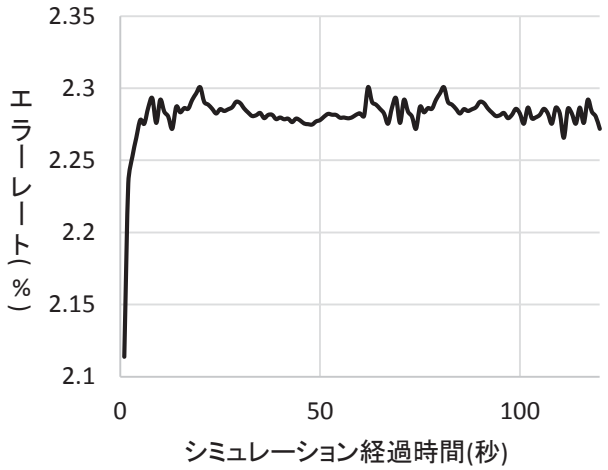

図 13 700Mhz 帯通信におけるエラーレート

6. まとめと今後の課題

本稿では，車車間通信により，狭域交通情報を効率的に周囲の車両で共有できるようにするため，安全運転支援システムで 700MHz 帯によって交換される周囲の車両の位置，速度，移動方向の情報に基づく順序付けにより中継転送方式を提案した．シミュレーションを通じて，提案方式を導入することで既存方式と比較して拡散率で最大 35.0%向上，遅延時間で最大 55.3%低減，通信回数を最大で 20.5%削減できていることがわかった．

また，中継優先度の割当車両台数を動的に変更することにより，通信トラヒックの様々な混雑度の状態において高い拡散率を維持できることがわかった．さらに交差道路の車両に中継優先度の割当てることにより，通信トラヒックの混雑時に優先度割当台数が低下した際でも交差道路の車両が中継することで拡散率が向上することがわかった．

今後の課題として，緊急度を持った情報の優先送信制御の導入，交通情報の特徴に基づいた配布範囲制御や銀座モデル等を利用した実環境に近いシミュレーションによる検証が挙げられる．

参考文献

[1] 宮本進生, 四方博之, ｼｬｸﾞﾀﾞﾙｵﾕｰﾝﾁﾒｸﾞ, ﾇﾘ ｼﾗｼﾞ ﾏﾊﾀﾞｯﾄﾞ, 大山 卓, 三浦 龍, 小花貞夫, CDMA vs. TDMA: 安全運転支援のための車車間通信システム特性評価, 電子情報通信学会論文誌, Vol.J93-A, No.7， pp.474-484 ,July 2010

[2] GokhanKorkmaz, EylemEkici , Fusun O zguner, Umit O zguner: “Urban Multi-Hop Broadcast Protocol for Inter-Vehicle Communication Systems,” in VANET '04 Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks , Pages 76-85 , 2004

[3] Chakkaphong Suthaputchakun, Aura Ganz,”Priority Based Inter-Vehicle Communication in Vehicular Ad-Hoc Networks using IEEE 802.11e,” in IEEE Vehicular Technology Conference, 2007.

[4] Chakkaphong Suthaputchakun, Zhili Sun, “Priority based Routing Protocol with Reliability Enhancement in Vehicular Ad hoc Network,” in The 2nd International Conference on Communications and Information Technology in 2012

[5] 姜巍, 川瀬悠,若山公威, 岩田彰, 白石善明 , 車車間通信におけるコリジョン数を削減するフラッディング方式の提案と評価 , 情報処理学会研究報告 , 2008(25), 51-57 , 2008-03-07

[6] Brad Williams, Tracy Camp”Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks”, MobiHoc '02 Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing,Pages 194 – 205

[7] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu ,“The Broadcast Storm Problem in a Mobile Ad Hoc Network,” in MobiCom '99 Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking

[8] Sang-woo Chang, Sang-sun Lee,” A Study on Distance-based Multi-hop Broadcast Scheme for Inter-Vehicle Communication”, IT Convergence and Security (ICITCS), 2013 International Conference on

[9] ARIB(電波産業界) STD-T109 , http://www.arib.or.jp/english/html/overview/doc/1-STD-T109v1_0.pdf

[10] Space-Time Engineering,Scenargie Simulator, https://www.spacetime-eng.com/en/

付録B. ソースコード

scensim_app_flooding.h

```
16  //吉川追加 define
17  #define MyEndTime 360
18  #define MyNextEventTime 1
19  #define MAX_RANGE 380
20  #define ROAD_WIDTH 15
21  #define OBSERVING_TIME 5
22  #define OBSERVING_START 1
23  #define NODES 500
24  #define MAX_PRIORITY 30
25  #define SPLIT_NUM 50
26  #define MAX_INTERSECTION_PRIORITY 15
27
28  //吉川追加 include
29  #include <map>
30  #include <utility>
31  #include <vector>
32
33
34  //Mac 用構造体
35  struct MacReceiveVector{
36      int nodeid;
37      int sourcenodeid;
38      char applicationid;
39      int sequencenumber;
40      int receivecount;
41
42      MacReceiveVector(
43          int initnodeid ,
44          int initsourcenodeid ,
45          char initapplicationid ,
46          int initsequencenumber ,
47          int initreceivecount
```

```
48          )
49          :
50      nodeid(initnodeid),
51          sourcenodeid(initsourcenodeid),
52          applicationid(initapplicationid),
53          sequencenumber(initsequencenumber),
54          receivecount(initreceivecount)
55      {}
56  };
57
58
59
60  struct NodePositionVector{
61      double nodepositionx;
62      double nodepositiony;
63      unsigned int nodeid;
64
65      NodePositionVector(
66          double initnodeposx,
67          double initnodeposy,
68          unsigned int initnodeid
69          )
70          :
71      nodepositionx(initnodeposx),
72          nodepositiony(initnodeposy),
73          nodeid(initnodeid)
74      {}
75  };
76
77
78  struct StatisticVector{
79      string applicationid;
80      unsigned int mynodeid;
81      unsigned int sourcenodeid;
82      unsigned int sequencenumber;
83      int pri;
```

```
84      int counter;
85      int getcounter;
86      int infotype;
87      int dissemination;
88      int infodirection;
89      int infoangle;
90      double sourcenodex;
91      double sourcenodey;
92      long long int generatetime;
93      unsigned int endtoenddelay;
94      int endtoenddelay_min;
95      int endtoenddelay_max;
96      int broadcastcount;
97      int receivecount;
98
99      StatisticVector(
100         string initapplicationid ,
101         unsigned int initmynodeid ,
102         unsigned int initsourcenodeid ,
103         unsigned int initsequencenumber ,
104         int initpri ,
105         int initcounter ,
106         int initgetcounter ,
107         int initinfotype ,
108         int initdissemination ,
109         int initinfodirection ,
110         int initinfoangle ,
111         double initsourcenodex ,
112         double initsourcenodey ,
113         long long int initgeneratetime ,
114         unsigned int initendtoenddelay ,
115         int initendtoenddelay_min ,
116         int initendtoenddelay_max ,
117         int initbroadcastcount ,
118         int initreceivecount
119         )
```

```
120         :
121     applicationid(initapplicationid),
122         mynodeid(initmynodeid),
123         sourcenodeid(initsourcenodeid),
124         sequencenumber(initsequencenumber),
125         pri(initpri),
126         counter(initcounter),
127         getcounter(initgetcounter),
128         infotype(initinfotype),
129         dissemination(initdissemination),
130         infodirection(initinfodirection),
131         infoangle(initinfoangle),
132         sourcenodex(initsourcenodex),
133         sourcenodey(initsourcenodey),
134         generatetime(initgeneratetime),
135         endtoenddelay(initendtoenddelay),
136         endtoenddelay_min(initendtoenddelay_min),
137         endtoenddelay_max(initendtoenddelay_max),
138         broadcastcount(initbroadcastcount),
139         receivecount(initreceivecount)
140     {}
141 };
142
143
144 //グローバル変数は appliction.cpp に記述
145 //吉川追加(統計用 Vector)
146 extern vector<StatisticVector> statvec;
147 extern vector<StatisticVector> rebroadcast_count;
148
149 //吉川追加(ロケーションマップ用 vector)
150 extern vector<NodePositionVector> posvec;
151
152 extern int Intersection_Priority;
153
154 //吉川追加(統計用変数)
155 extern int totalgetcounter;
```

```
156 extern int totalcounter;
157 extern int diff_distribution[11];
158 extern int delay_distribution[11];
159 extern int generate_infos;
160
161 //PHY 操作
162 extern int total_error_phy;
163 extern int total_success_phy;
164 extern int error_noise_phy;
165 extern int error_duringtransmission_phy;
166 extern int error_tooweak_phy;
167 extern int error_signalscaptured_phy;
168 extern int cancel_on_mac;
169 extern int total_busy_time;
170 extern int total_idle_time;
171 extern int total_send_mac;
172
173 extern int interfering_count;
174 extern int weaksig_count;
175 extern int signalduring_count;
176
177 extern int CorruptedFrame;
178 extern int RebroadCastTimes;
179 extern int SignalCaptureTimes;
180 extern int FramesWithErros;
181 extern int FramesTransmitted;
182 extern int FramesReceived;
183
184 extern double total_busy_time_ms;
185 extern double total_idle_time_ms;
186
187 //<-------吉川追加(統計用構造体)
188
189
190 namespace ScenSim {
191
```

```
192     using std::cerr;
193     using std::endl;
194
195
196     //--------------------------------------------------------------------------------------------------
197
198     class FloodingApplication: public Application , public enable_shared_from_this<FloodingApplication> {
199     public:
200         static const string modelName;
201         static const int APPLICATION_ID_CHAR_MAX_LENGTH = 16;
202         typedef char ApplicationIdCharType[APPLICATION_ID_CHAR_MAX_LENGTH];
203
204         //int NewPrioritySolver(int sourcenodeid , int mynodeid);
205         int AppPrioritySolver(unsigned int sourcenodeid , unsigned int mynodeid);
206         int AppPrioritySolver_InterSection(unsigned int sourcenodeid , unsigned int mynodeid);
207         int PrioritySolver_SPLIT(int nodeid , double sourcex , double sourcey);
208
209         int PrioritySolver(int nodeid , double sourcex , double sourcey);
210
211         //吉川追加(統計用)--->
212         void initStatistic();
213         void StatisticSetting();
214         void Statistic(ApplicationIdType applicationId);
215         //<---
216
351
352         //吉川追加(統計用)--->
353         class initStatisticEvent: public SimulationEvent {
354         public:
355             explicit
356                 initStatisticEvent(
357                 const shared_ptr<FloodingApplication>& initFloodingApplicationPtr ,
358                 const ApplicationIdType& initApplicationId)
359                 :
360             floodingApplicationPtr(initFloodingApplicationPtr)
361             {}
```

```
362
363             virtual void ExecuteEvent()
364             {
365                 floodingApplicationPtr->initStatistic();
366             }
367
368         private:
369             shared_ptr<FloodingApplication> floodingApplicationPtr;
370             ApplicationIdType applicationId;
371         };
372
373
374         class StatisticEvent: public SimulationEvent {
375         public:
376             explicit
377                 StatisticEvent(
378                 const shared_ptr<FloodingApplication>& initFloodingApplicationPtr ,
379                 const ApplicationIdType& initApplicationId)
380                 :
381             floodingApplicationPtr(initFloodingApplicationPtr) ,
382                 applicationId(initApplicationId)
383
384             {}
385
386             virtual void ExecuteEvent()
387             {
388                 floodingApplicationPtr->Statistic(applicationId);
389             }
390
391         private:
392             shared_ptr<FloodingApplication> floodingApplicationPtr;
393             ApplicationIdType applicationId;
394         };
395         //<---統計用
396
635         //吉川追加(統計用 Vector)
```

```
636         //vector<StatisticVector> statvec_get;
637         //vector<StatisticVector> rebroadcast_count;
638 
639         bool useVirtualPayload;
640 
641         int finish_stats;
642         int start_stats;
643 
644     };//FloodingApplication//
645 
646 
647     inline
648         FloodingApplication::FloodingApplication(
649         const ParameterDatabaseReader& initParameterDatabaseReader ,
650         const shared_ptr<SimulationEngineInterface>& initSimulationEngineInterfacePtr ,
651         const ApplicationIdType& initApplicationId ,
652         const NodeIdType& initNodeId ,
653         const unsigned short int initDefaultApplicationPortId)
654         :
655     Application(initSimulationEngineInterfacePtr ,  modelName) ,
656         nodeId(initNodeId) ,
657         destinationPortId(initDefaultApplicationPortId) ,
658         packetHandlerPtr() ,
659         floodingSenderSettings() ,
660         floodingSenderStats() ,
661         floodingReceiverStats() ,
662         countOfPacketReceived() ,
663         minDistanceBetweenNodesInMeters() ,
664         useVirtualPayload(false)
665     {
666         if (initParameterDatabaseReader.ParameterExists(
667             "flooding-auto-port-mode" ,  nodeId ,  initApplicationId)) {
668 
669                 if (!initParameterDatabaseReader.ReadBool(
670                     "flooding-auto-port-mode" ,  nodeId ,  initApplicationId)) {
671 
```

```
672                         destinationPortId = initParameterDatabaseReader.ReadInt(
673                             "flooding-destination-port", nodeId, initApplicationId);
674                 }//if//
675         }//if//
676 
677         if (initParameterDatabaseReader.ParameterExists(
678             "flooding-use-virtual-payload", nodeId, initApplicationId)) {
679 
680                 useVirtualPayload = initParameterDatabaseReader.ReadBool(
681                     "flooding-use-virtual-payload", nodeId, initApplicationId);
682         }//if//
683 
684         finish_stats = 0;
685         start_stats = 0;
686 
687     }//FloodingApplication//
688 
689 
690     inline
691         void FloodingApplication::CompleteInitialization()
692     {
693         packetHandlerPtr = shared_ptr<PacketHandler>(new PacketHandler(shared_from_this()));
694 
695         assert(transportLayerPtr->udpPtr->PortIsAvailable(destinationPortId));
696 
697         transportLayerPtr->udpPtr->OpenSpecificUdpPort(
698             NetworkAddress::anyAddress,
699             destinationPortId,
700             packetHandlerPtr);
701 
702     }//CompleteInitialization//
703 
704 
705     inline
706         void FloodingApplication::AddSenderSetting(
707         const ParameterDatabaseReader& theParameterDatabaseReader,
```

```
708         const ApplicationIdType& initApplicationId)
709     {
710         applicationId = initApplicationId;
711 
712         const int packetPayloadSizeBytes =
713             theParameterDatabaseReader.ReadInt("flooding-payload-size-bytes", nodeId, applicationId);
714 
715         const TimeType packetInterval =
716             theParameterDatabaseReader.ReadTime("flooding-interval", nodeId, applicationId);
717 
718         const TimeType floodingStartTime =
719             theParameterDatabaseReader.ReadTime("flooding-start-time", nodeId, applicationId);
720 
721         const TimeType floodingEndTime =
722             theParameterDatabaseReader.ReadTime("flooding-end-time", nodeId, applicationId);
723 
724         const int floodingPriority =
725             theParameterDatabaseReader.ReadInt("flooding-priority", nodeId, applicationId);
726 
727         const unsigned int maxHopCount =
728             theParameterDatabaseReader.ReadInt("flooding-max-hop-count", nodeId, applicationId);
729 
730         const TimeType minWaitingPeriod =
731             theParameterDatabaseReader.ReadTime("flooding-min-waiting-period", nodeId, applicationId);
732 
733         const TimeType maxWaitingPeriod =
734             theParameterDatabaseReader.ReadTime("flooding-max-waiting-period", nodeId, applicationId);
735 
736         const unsigned int counterThreshold =
737             theParameterDatabaseReader.ReadInt("flooding-counter-threshold", nodeId, applicationId);
738 
739         const double distanceThresholdInMeters =
740             theParameterDatabaseReader.ReadDouble("flooding-distance-threshold-in-meters", nodeId,
    applicationId);
741 
```

```
742         int mypriority = 0;
743         unsigned int myRelayNodeId = 0;
744 
745         if (packetPayloadSizeBytes < (int)sizeof(FloodingPayloadType)) {
746             cerr << "Error: Packet payload size ("
747                 << packetPayloadSizeBytes << ") should be "
748                 << sizeof(FloodingPayloadType) << " bytes or larger." << endl;
749             exit(1);
750         }//if//
751 
752         if (packetInterval <= ZERO_TIME) {
753             cerr << "Error: Broadcast interval ("
754                 << minWaitingPeriod << ") should be larger than "
755                 << ZERO_TIME << "." << endl;
756             exit(1);
757         }//if//
758 
759         if (floodingStartTime < ZERO_TIME) {
760             cerr << "Error: Start time ("
761                 << floodingStartTime << ") should be "
762                 << ZERO_TIME << " or larger." << endl;
763             exit(1);
764         }//if//
765 
766         if (floodingStartTime >= floodingEndTime) {
767             cerr << "Error: End time ("
768                 << floodingEndTime << ") should be larger than start time ("
769                 << floodingStartTime << ")." << endl;
770             exit(1);
771         }//if//
772 
773         if (minWaitingPeriod < ZERO_TIME) {
774             cerr << "Error: Min waiting period ("
775                 << minWaitingPeriod << ") should be "
776                 << ZERO_TIME << " or larger." << endl;
777             exit(1);
```

```
778         }//if//
779
780         if (minWaitingPeriod > maxWaitingPeriod) {
781             cerr << "Error: Max waiting period ("
782                 << maxWaitingPeriod << ") should be min waiting period ("
783                 << minWaitingPeriod << ") or larger." << endl;
784             exit(1);
785         }//if//
786
787         if (distanceThresholdInMeters < 0) {
788             cerr << "Error: Distance threshold ("
789                 << distanceThresholdInMeters << ") should be 0 meters or larger." << endl;
790             exit(1);
791         }//if//
792
793         FloodingSenderSettingType floodingSenderSetting(
794             packetPayloadSizeBytes ,
795             packetInterval ,
796             floodingEndTime ,
797             floodingPriority ,
798             maxHopCount ,
799             minWaitingPeriod ,
800             maxWaitingPeriod ,
801             counterThreshold ,
802             distanceThresholdInMeters ,
803             mypriority ,
804             myRelayNodeId);
805
806         floodingSenderSettings.insert(make_pair(applicationId ,  floodingSenderSetting));
807
808         FloodingSenderStatType floodingSenderStat(
809             simulationEngineInterfacePtr ,
810             modelName ,
811             applicationId);
812
813         floodingSenderStats.insert(make_pair(applicationId ,  floodingSenderStat));
```

```
814
815         const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
816
817         TimeType startTime = floodingStartTime;
818
819         if (theParameterDatabaseReader.ParameterExists(
820             "flooding-start-time-max-jitter" , nodeId , applicationId)) {
821
822                 const TimeType maxStartTimeJitter =
823                     theParameterDatabaseReader.ReadTime(
824                     "flooding-start-time-max-jitter" , nodeId , applicationId);
825
826                 startTime += static_cast<TimeType>(
827                     aRandomNumberGeneratorPtr->GenerateRandomDouble() * maxStartTimeJitter);
828         }//if//
829
830         if (currentTime > startTime) {
831             const size_t nextTransmissionTime =
832                 size_t(ceil(double(currentTime - floodingStartTime) / packetInterval));
833             startTime += nextTransmissionTime * packetInterval;
834         }//if//
835
836         if (startTime < floodingEndTime) {
837             simulationEngineInterfacePtr->ScheduleEvent(
838                 unique_ptr<SimulationEvent>(new FloodingStartEvent(shared_from_this() , applicationId)) ,
839                 startTime);
840         }//if//
841
842     }//AddSenderSetting//
843
844
845     inline
846         void FloodingApplication::AddReceiverSetting(
847         const ApplicationIdType& senderApplicationId)
848     {
849         FloodingReceiverStatType floodingReceiverStat(
```

```
850             simulationEngineInterfacePtr ,
851             modelName ,
852             senderApplicationId);
853 
854         floodingReceiverStats.insert(
855             make_pair(senderApplicationId , floodingReceiverStat));
856 
857     }//AddReceiverSetting//
858 
859     //吉川追加(統計用)--->
860     inline
861         void FloodingApplication::StatisticSetting(){
862             const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
863             TimeType nextPacketTime = currentTime + MyNextEventTime * SECOND;
864             TimeType floodingEndTime = MyEndTime * SECOND;
865 
866             EventRescheduleTicket* myEventTicket1 = new EventRescheduleTicket();
867             shared_ptr<StatisticEvent> fseptr1;
868 
869             fseptr1.reset(new StatisticEvent(shared_from_this() , applicationId));
870 
871             if (nextPacketTime <= floodingEndTime) {
872                 simulationEngineInterfacePtr->ScheduleEvent(
873                     fseptr1 ,
874                     nextPacketTime ,
875                     *myEventTicket1);
876             }
877     }
878 
879     inline
880         void FloodingApplication::initStatistic(){
881 
882             const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
883             TimeType nextPacketTime = currentTime + MyNextEventTime * SECOND;
884             TimeType floodingEndTime = MyEndTime * SECOND;
885 
```

```
886             //変数
887             double total_diff = 0;
888             double currentTime_output = (double)((double)currentTime/(double)SECOND);
889             double total_delay = 0;
890             double avg_delay = 0;
891             double total_errrate = 0;
892             double min_diff = 1000;
893             double max_diff = -1;
894             double min_delay = 1 * SECOND;
895             double max_delay = -1;
896
897             int temp = 0;
898             double diff_current = 0;
899             double delay_current = 0;
900
901             //初期設定
902             if(start_stats == 0){
903                 start_stats++;
904                 memset(diff_distribution , 0 ,sizeof(diff_distribution));
905                 memset(delay_distribution , 0 , sizeof(delay_distribution));
906                 cout << "経過時間,情報生成数,情報取得数,情報取得上限,拡散率,最高拡散率,最低拡散率,平均遅延(ms),最大遅延(ms),最小遅延(ns),総送信回数,中継回数,受信成功数,(MAC層)破損ﾌﾚｰﾑ数,ｴﾗｰﾚｰﾄ,(PHY層)ﾌﾚｰﾑ受信ｴﾗｰ数,(PHY)ﾌﾚｰﾑ送信,(PHY)ﾌﾚｰﾑ受信数,干渉波検出数,(ｴﾗｰ)送信中受信,(ｴﾗｰ)微弱信号,(ｴﾗｰ)ｼｸﾞﾅﾙｷｬﾌﾟﾁｬ,BusyTime(ms)(Node1),IdleTime(ms)(Node1),チャネル使用率" << endl;
907
908             }
909
910             //計算処理
911             vector<StatisticVector>::iterator statvecitr = statvec.begin();
912             while(statvecitr != statvec.end()){
913                 //if(((*statvecitr).generatetime > currentTime - (OBSERVING_TIME * SECOND)) && (currentTime - (*statvecitr).generatetime > (OBSERVING_START * SECOND))){
914                 if(( currentTime - (*statvecitr).generatetime < (OBSERVING_TIME * SECOND)) && (currentTime - (*statvecitr).generatetime > (OBSERVING_START * SECOND))){
915                     totalcounter = totalcounter + NODES;
916                     totalgetcounter = totalgetcounter + (*statvecitr).receivecount;
```

```
917                     total_delay = total_delay + (*statvecitr).endtoenddelay;
918
919                     diff_current = (double)((double)(*statvecitr).receivecount * 100.0f / (double)NODES );
920                     temp = (int)(diff_current / 10);
921                     diff_distribution[temp] = diff_distribution[temp] + 1;
922                     /*
923                     delay_current = (*statvecitr).endtoenddelay;
924                     temp = (int)((*statvecitr).endtoenddelay/MILLI_SECOND);
925                     if(temp < 11){
926                         delay_distribution[temp] = delay_distribution[temp] + 1;
927                     }
928                     else{
929                         delay_distribution[10] = delay_distribution[10] + 1;
930                     }
931                     */
932
933                     //最大拡散率
934                     if(max_diff < diff_current){
935                         max_diff = diff_current;
936                     }
937                     //最低拡散率
938                     if(min_diff > diff_current){
939                         min_diff = diff_current;
940                     }
941
942                     //最大遅延
943                     if(max_delay < (*statvecitr).endtoenddelay_max){
944                         max_delay = (*statvecitr).endtoenddelay_max;
945                     }
946
947                     //最低遅延
948                     if(min_delay >(*statvecitr).endtoenddelay_min){
949                         min_delay = (*statvecitr).endtoenddelay_min;
950                     }
951                     ++statvecitr;
952                 }
```

```
953                 else if((currentTime - (*statvecitr).generatetime) > (OBSERVING_TIME * SECOND)){
954                     statvecitr = statvec.erase(statvecitr);
955                 }
956                 else{
957                     ++statvecitr;
958                 }
959             }
960 
961             //拡散率と遅延時間の計算
962             if(totalcounter != 0){
963                 total_diff = double((double)totalgetcounter / (double)totalcounter * 100.0f);
964                 if(totalgetcounter != 0){
965                 avg_delay = double((double)total_delay / (double)totalgetcounter);
966                 }
967             }
968 
969             if(total_error_phy != 0 &&total_success_phy != 0){
970                 total_errrate = double((double)total_error_phy / ((double)total_error_phy + (double)total_success_phy) * 100.0f);
971             }
972 
973             //出力
974             if(min_diff == 1000){
975                 cout << currentTime_output << " , "
976                     << generate_infos << " , "
977                     << totalgetcounter << " , "
978                     << totalcounter << " , "
979                     << "N/A" << " , "
980                     << "N/A" << " , "
981                     << "N/A" << " , "
982                     << "N/A" << " , "
983                     << "N/A"<< " , "
984                     << "N/A"<< " , "
985                     << total_send_mac<< " , "
986                     << RebroadCastTimes<< " , "
987                     << total_success_phy << " , "
```

```
988                     << CorruptedFrame << " , "
989                     << "N/A" << " , "
990                     << FramesWithErros << " , "
991                     << FramesTransmitted << " , "
992                     << FramesReceived << " , "
993                     << interfering_count << " , "
994                     << signalduring_count << " , "
995                     << weaksig_count << " , "
996                     << SignalCaptureTimes << " , "
997                     << total_busy_time_ms << " , "
998                     << total_idle_time_ms << " , "
999                     << "N/A" << " , "
1000                    <<endl;
1001            }
1002
1003            else{
1004                cout << currentTime_output << " , "
1005                    << generate_infos << " , "
1006                    << totalgetcounter << " , "
1007                    << totalcounter << " , "
1008                    << total_diff << " , "
1009                    << max_diff << " , "
1010                    << min_diff << " , "
1011                    << avg_delay << " , "
1012                    << max_delay / MILLI_SECOND<< " , "
1013                    << min_delay / NANO_SECOND<< " , "
1014                    << total_send_mac<< " , "
1015                    << RebroadCastTimes<< " , "
1016                    << total_success_phy << " , "
1017                    << CorruptedFrame << " , "
1018                    << (double)CorruptedFrame / (double)(total_success_phy + CorruptedFrame) * 100.0f << " ,
"
1019                    << FramesWithErros << " , "
1020                    << FramesTransmitted << " , "
1021                    << FramesReceived << " , "
1022                    << interfering_count << " , "
```

```
1023                    << signalduring_count << " , "
1024                    << weaksig_count << " , "
1025                    << SignalCaptureTimes << " , "
1026                    << total_busy_time_ms << " , "
1027                    << total_idle_time_ms << " , "
1028                    << (double)total_busy_time_ms / (double)(total_busy_time_ms + total_idle_time_ms) * 100.0f
    << " , "
1029                    <<endl;
1030            }
1031
1032            //拡散率分布
1033            /*
1034            if(max_diff != -1){
1035                for(int i = 0 ; i < 11 ; i++){
1036                    if(i == 10){
1037                        cout << i << " - " << i + 9 << "% , " << diff_distribution[i] << endl;
1038                    }
1039                    else{
1040                        cout << i << " - " << i + 9 << "% , " << diff_distribution[i] << cout.flush();
1041                    }
1042                }
1043            }
1044            */
1045
1046            /*メモ
1047            << total_errrate << " , "
1048            << total_error_phy << " , "
1049            << cancel_on_mac << " , "
1050            << total_busy_time<< " , "
1051            << total_idle_time<< " , "
1052            << interfering_count<< " , "
1053            << weaksig_count<< " , "
1054            << signalduring_count
1055            */
1056
1057            //初期化
```

```
1058            totalcounter = 0;
1059            totalgetcounter = 0;
1060
1061            //中継車両数変数
1062            int rebroadcast_totalcount = 0;
1063
1064            //中継車両数表示
1065            if(currentTime_output == 120 && finish_stats == 0){
1066
1067                cout << endl;
1068
1069                //拡散率分布表示
1070                for(int i = 0 ; i < 11 ; i++){
1071                    if(i == 0){
1072                    cout << "拡散率 , 0 - 9% ,";
1073                    }
1074                    else if( i == 10 ){
1075                        cout << "100%" << endl;
1076                    }
1077                    else{
1078                        cout << i * 10 << " - " << i * 10 + 9 << "% ,";
1079                    }
1080                }
1081                //拡散率分布表示
1082                for(int i = 0 ; i < 11 ; i++){
1083                    if(i == 0){
1084                    cout << " ," << diff_distribution[i] << " ,";
1085                    }
1086                    else if( i == 10 ){
1087                        cout << diff_distribution[i] << endl;
1088                    }
1089                    else{
1090                        cout << diff_distribution[i] << " ,";
1091                    }
1092                }
1093
```

```
1094                cout << endl;
1095
1096                //遅延時間分布表示
1097                //拡散率分布表示
1098                for(int i = 0 ; i < 11 ; i++){
1099                    if(i == 0){
1100                    cout << "遅延時間 , 0 - 99ms ,";
1101                    }
1102                    else if( i == 10 ){
1103                        cout << "1000ms - " << endl;
1104                    }
1105                    else{
1106                        cout << i * 100 << " - " << i * 100 + 99 << "ms ,";
1107                    }
1108                }
1109                //拡散率分布表示
1110                for(int i = 0 ; i < 11 ; i++){
1111                    if(i == 0){
1112                    cout << " ," << delay_distribution[i] << " ,";
1113                    }
1114                    else if( i == 10 ){
1115                        cout << delay_distribution[i] << endl;
1116                    }
1117                    else{
1118                        cout << delay_distribution[i] << " ,";
1119                    }
1120                }
1121
1122                cout << endl;
1123
1124                vector<StatisticVector>::iterator rebroadcast_countitr = rebroadcast_count.begin();
1125                while(rebroadcast_countitr != rebroadcast_count.end()){
1126                    cout << "[APPID:SEQNUM = "<< (*rebroadcast_countitr).applicationid << ":" <<
(*rebroadcast_countitr).sequencenumber << "] ReceiveCount = " << (*rebroadcast_countitr).receivecount << endl;
1127                    rebroadcast_totalcount += (*rebroadcast_countitr).receivecount;
1128                    ++rebroadcast_countitr;
```

```
1129                }
1130                cout << "TOTAL rebroadcast_times = " << rebroadcast_totalcount << endl;
1131                finish_stats++;
1132            }
1133
1134            EventRescheduleTicket* myEventTicket1 = new EventRescheduleTicket();
1135            shared_ptr<initStatisticEvent> fseptr1;
1136
1137            fseptr1.reset(new initStatisticEvent(shared_from_this()  ,  applicationId));
1138
1139            if (nextPacketTime <= floodingEndTime) {
1140                simulationEngineInterfacePtr->ScheduleEvent(
1141                    fseptr1 ,
1142                    nextPacketTime ,
1143                    *myEventTicket1);
1144            }
1145    }
1146
1147    //Statistic()は 1 ノードがそれぞれ実行する
1148    //位置情報の更新も Statistic()に任せる
1149    //実行回数=ノード数
1150    //statvec は全体の情報を保持
1151
1152    inline
1153        void FloodingApplication::Statistic(ApplicationIdType applicationId)
1154    {
1155        ObjectMobilityPosition nodePosition;
1156        nodeMobilityModelPtr->GetPositionForTime(
1157            simulationEngineInterfacePtr->CurrentTime() ,  nodePosition);
1158
1159        const double nodePositionX = nodePosition.X_PositionMeters();
1160        const double nodePositionY = nodePosition.Y_PositionMeters();
1161
1162        const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
1163
1164        vector<StatisticVector>::iterator statvecit = statvec.begin();
```

```
1165
1166        TimeType nextPacketTime = currentTime + MyNextEventTime * SECOND;
1167        TimeType floodingEndTime = MyEndTime * SECOND;
1168
1169        EventRescheduleTicket* myEventTicket1 = new EventRescheduleTicket();
1170        shared_ptr<StatisticEvent> fseptr1;
1171
1172        fseptr1.reset(new StatisticEvent(shared_from_this() , applicationId));
1173
1174        if (nextPacketTime <= floodingEndTime) {
1175            simulationEngineInterfacePtr->ScheduleEvent(
1176                fseptr1 ,
1177                nextPacketTime ,
1178                *myEventTicket1);
1179        }
1180
1181        //以下ロケーションマップ作成
1182        vector<NodePositionVector>::iterator posvecitr = posvec.begin();
1183
1184        int exist_check = 0;
1185
1186        while(posvecitr != posvec.end()){
1187
1188            if((*posvecitr).nodeid == nodeId){
1189                exist_check = 1;
1190                (*posvecitr).nodepositionx = nodePositionX;
1191                (*posvecitr).nodepositiony = nodePositionY;
1192                break;
1193            }
1194            ++posvecitr;
1195        }
1196
1197        if(exist_check == 0){
1198            posvec.push_back(NodePositionVector(nodePositionX , nodePositionY , nodeId));
1199        }
1200    }
```

```
1201    //<---統計用
1202
1203    inline
1204        void FloodingApplication::Broadcast(const ApplicationIdType& applicationId)
1205    {
1206        typedef map<ApplicationIdType , FloodingSenderSettingType>::iterator IterType;
1207        IterType iter = floodingSenderSettings.find(applicationId);
1208
1209        assert(iter != floodingSenderSettings.end());
1210        assert((*iter).second.sequenceNumber < UINT_MAX);
1211        (*iter).second.sequenceNumber++;
1212
1213        ObjectMobilityPosition nodePosition;
1214        nodeMobilityModelPtr->GetPositionForTime(
1215            simulationEngineInterfacePtr->CurrentTime() , nodePosition);
1216
1217        const int packetPayloadSizeBytes = (*iter).second.packetPayloadSizeBytes;
1218        const TimeType packetInterval = (*iter).second.packetInterval;
1219        const TimeType floodingEndTime = (*iter).second.floodingEndTime;
1220        int sequenceNumber = (*iter).second.sequenceNumber;
1221        const PacketPriorityType floodingPriority = (*iter).second.floodingPriority;
1222        const unsigned int maxHopCount = (*iter).second.maxHopCount;
1223        const TimeType minWaitingPeriod = (*iter).second.minWaitingPeriod;
1224        const TimeType maxWaitingPeriod = (*iter).second.maxWaitingPeriod;
1225        const unsigned int counterThreshold = (*iter).second.counterThreshold;
1226        const double distanceThresholdInMeters = (*iter).second.distanceThresholdInMeters;
1227        const int mypriority = 0;
1228        unsigned int myRelayNodeId = 0;
1229        double nodePositionX = nodePosition.X_PositionMeters();
1230        double nodePositionY = nodePosition.Y_PositionMeters();
1231        const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
1232        //const TimeType nextPacketTime = currentTime + packetInterval + (nodeId * 100000000);
1233        //const TimeType nextPacketTime = currentTime + packetInterval + (nodeId * MILLI_SECOND);
1234        //const TimeType nextPacketTime = currentTime + (packetInterval / NODES) * nodeId;
1235
1236        int randnum = aRandomNumberGeneratorPtr->GenerateRandomInt( 0 , 100000);
```

```
        const TimeType nextPacketTime = currentTime + (double)((double)packetInterval * (double)randnum /
1237    (double)50000);
1238
1239
1240
1241        /*
1242        TimeType nextPacketTime = 0;
1243        if(sequenceNumber == 1){
1244            nextPacketTime = currentTime + (double)((double)packetInterval / (double)NODES);
1245        }
1246        else{
1247            nextPacketTime = currentTime + packetInterval;
1248        }
1249        */
1250
1251        FloodingPayloadType floodingPayload(
1252            applicationId ,
1253            nodeId ,
1254            sequenceNumber ,
1255            currentTime ,
1256            nodePositionX ,
1257            nodePositionY ,
1258            nodePositionX ,
1259            nodePositionY ,
1260            floodingPriority ,
1261            maxHopCount ,
1262            minWaitingPeriod ,
1263            maxWaitingPeriod ,
1264            counterThreshold ,
1265            distanceThresholdInMeters ,
1266            mypriority ,
1267            myRelayNodeId);
1268
1269        floodingPayload.mypriority = 0;
1270        floodingPayload.myRelayNodeId = nodeId;
1271
```

```
1272         unique_ptr<Packet> packetPtr =
1273             Packet::CreatePacket(
1274             *simulationEngineInterfacePtr ,
1275             floodingPayload ,
1276             packetPayloadSizeBytes ,
1277             useVirtualPayload);
1278 
1279         OutputTraceAndStatsForBroadcast(
1280             applicationId ,
1281             sequenceNumber ,
1282             packetPtr->GetPacketId() ,
1283             packetPtr->LengthBytes());
1284 
1285         transportLayerPtr->udpPtr->SendPacket(
1286             packetPtr , 0 , NetworkAddress::broadcastAddress , destinationPortId , floodingPriority);
1287 
1288         //初期データ挿入
1289         statvec.push_back(StatisticVector(applicationId , 0 , nodeId , sequenceNumber , 0 , 0 , 0 , 0 , 0 , 0 , 0 , nodePositionX , nodePositionY , currentTime , 0 , 1 * SECOND , 0 , 0 , 1));
1290         rebroadcast_count.push_back(StatisticVector(applicationId , 0 , nodeId , sequenceNumber , 0 , 0 , 0 , 0 , 0 , 0 , 0 , nodePositionX , nodePositionY , currentTime , 0 , 0 , 0 , 0 , 1));
1291 
1292         generate_infos++;
1293 
1294         if (nextPacketTime < floodingEndTime) {
1295             simulationEngineInterfacePtr->ScheduleEvent(
1296                 unique_ptr<SimulationEvent>(new FloodingStartEvent(shared_from_this() , applicationId)) ,
1297                 nextPacketTime);
1298         }//if//
1299     }//Broadcast//


1416         int FloodingApplication::AppPrioritySolver(unsigned int sourcenodeid , unsigned int mynodeid){
1417             int distcheck = 0;
1418             int directioncheck = 0;
1419             int prioritynum = 0;
1420             double sourcex = 0;
```

```
1421            double sourcey = 0;
1422            double mydistance = 0;
1423            double otherdistance = 0;
1424
1425            vector<NodePositionVector>::iterator posvecitr_new = posvec.begin();
1426            vector<NodePositionVector>::iterator posvecitr = posvec.begin();
1427            vector<NodePositionVector>::iterator posvecitr2 = posvec.begin();
1428
1429            while(posvecitr_new != posvec.end()){
1430                if((*posvecitr_new).nodeid == sourcenodeid){
1431                    sourcex = (*posvecitr_new).nodepositionx;
1432                    sourcey = (*posvecitr_new).nodepositiony;
1433                    break;
1434                }
1435                ++posvecitr_new;
1436            }
1437            //sqrt(((*posvecitr).nodepositionx - sourcex) * ((*posvecitr).nodepositionx - sourcex) +
((*posvecitr).nodepositiony - sourcey) * ((*posvecitr).nodepositiony - sourcey));
1438
1439            while(posvecitr != posvec.end()){
1440                if((*posvecitr).nodeid == mynodeid){
1441                    mydistance = sqrt(((*posvecitr).nodepositionx - sourcex) * ((*posvecitr).nodepositionx -
sourcex) + ((*posvecitr).nodepositiony - sourcey) * ((*posvecitr).nodepositiony - sourcey));
1442                    break;
1443                }
1444                ++posvecitr;
1445            }
1446
1447            if(abs((*posvecitr).nodepositionx - sourcex) > ROAD_WIDTH && abs((*posvecitr).nodepositiony -
sourcey > ROAD_WIDTH)){
1448                return MAX_PRIORITY;
1449            }
1450
1451            //cout << "[APPLayer]PrioritySolver MyDistance = " << mydistance << " :: NodeId = " << nodeId <<
endl;
1452
```

```
1453            while(posvecitr2 != posvec.end()){
                otherdistance = sqrt(((*posvecitr2).nodepositionx - sourcex) * ((*posvecitr2).nodepositionx
1454 - sourcex) + ((*posvecitr2).nodepositiony - sourcey) * ((*posvecitr2).nodepositiony - sourcey));
1455
1456                if(otherdistance < MAX_RANGE){
1457                    if(otherdistance > mydistance){
                        if(abs((*posvecitr2).nodepositionx - sourcex) < ROAD_WIDTH ||
1458 abs((*posvecitr2).nodepositiony - sourcey < ROAD_WIDTH)){
                            if(abs((*posvecitr2).nodepositionx - sourcex) > ROAD_WIDTH &&
1459 abs((*posvecitr2).nodepositiony - sourcey > ROAD_WIDTH)){
1460                            }
1461                            else{
1462                                priorityнum++;
1463                            }
1464                        }
1465                    }
1466                    else if(otherdistance == mydistance && (*posvecitr2).nodeid > mynodeid){
1467                        priorityнum++;
1468                    }
1469                }
1470                ++posvecitr2;
1471            }
1472
1473            if(priorityнum > MAX_PRIORITY){
1474                //cout << "return priority = " << MAX_PRIORITY << endl;
1475                return MAX_PRIORITY;
1476            }
1477            //cout << "return priority = " << priorityнum << endl;
1478            return priorityнum;
1479    }
1480
1481    inline
        int FloodingApplication::AppPrioritySolver_InterSection(unsigned int sourcenodeid , unsigned int
1482 mynodeid){
1483            int distcheck = 0;
1484            int directioncheck = 0;
```

```
1485            int prioritynum = 0;
1486            double sourcex = 0;
1487            double sourcey = 0;
1488            double mydistance = 0;
1489            double otherdistance = 0;
1490
1491            vector<NodePositionVector>::iterator posvecitr_new = posvec.begin();
1492            vector<NodePositionVector>::iterator posvecitr = posvec.begin();
1493
1494            while(posvecitr_new != posvec.end()){
1495                if((*posvecitr_new).nodeid == sourcenodeid){
1496                    sourcex = (*posvecitr_new).nodepositionx;
1497                    sourcey = (*posvecitr_new).nodepositiony;
1498                    break;
1499                }
1500                ++posvecitr_new;
1501            }
1502            //sqrt(((*posvecitr).nodepositionx - sourcex) * ((*posvecitr).nodepositionx - sourcex) +
((*posvecitr).nodepositiony - sourcey) * ((*posvecitr).nodepositiony - sourcey));
1503
1504            while(posvecitr != posvec.end()){
1505                if((*posvecitr).nodeid == mynodeid){
1506                    if(abs((*posvecitr).nodepositionx - sourcex) > ROAD_WIDTH &&
abs((*posvecitr).nodepositiony - sourcey > ROAD_WIDTH)){
1507                        return 1;
1508                    }
1509                }
1510                ++posvecitr;
1511            }
1512            return 0;
1513    }
1514
1545
1546        inline
1547            void FloodingApplication::Rebroadcast(
1548            FloodingPayloadType& floodingPayload ,
```

```
1549            const unsigned int packetPayloadSizeBytes)
1550        {
1551            const PacketPriorityType floodingPriority = floodingPayload.floodingPriority;
1552
1553            ObjectMobilityPosition nodePosition;
1554            nodeMobilityModelPtr->GetPositionForTime(
1555                simulationEngineInterfacePtr->CurrentTime() , nodePosition);
1556
1557            //int temp = 0;
1558            //temp = PrioritySolver(nodeId , floodingPayload.nodePositionX , floodingPayload.nodePositionY);
1559            //floodingPayload.mypriority = temp;
1560
1561            floodingPayload.nodePositionX = nodePosition.X_PositionMeters();
1562            floodingPayload.nodePositionY = nodePosition.Y_PositionMeters();
1563
1564            vector<StatisticVector>::iterator rebroadcast_countitr = rebroadcast_count.begin();
1565            while(rebroadcast_countitr != rebroadcast_count.end()){
1566                if((*rebroadcast_countitr).sourcenodeid == floodingPayload.id.sourceNodeId &&
(*rebroadcast_countitr).sequencenumber == floodingPayload.id.sequenceNumber){
1567                    (*rebroadcast_countitr).receivecount++;
1568                    break;
1569                }
1570                ++rebroadcast_countitr;
1571            }
1572
1573            RebroadCastTimes++;
1574
1575            unique_ptr<Packet> packetPtr =
1576                Packet::CreatePacket(
1577                *simulationEngineInterfacePtr ,
1578                floodingPayload ,
1579                packetPayloadSizeBytes ,
1580                useVirtualPayload);
1581
1582            OutputTraceAndStatsForRebroadcast(
1583                string(floodingPayload.id.applicationIdChar) ,
```

```
1584                floodingPayload.id.sequenceNumber ,
1585                packetPtr->GetPacketId() ,
1586                packetPtr->LengthBytes());
1587
1588            transportLayerPtr->udpPtr->SendPacket(
1589                packetPtr , 0 , NetworkAddress::broadcastAddress , destinationPortId , floodingPriority);
1590
1591        }//Rebroadcast//
1592
1593
1594
1595        inline
1596            int FloodingApplication::PrioritySolver(int nodeid , double sourcex , double sourcey)
1597        {
1598            int distcheck = 0;
1599            int directioncheck = 0;
1600            int prioritynum = 0;
1601            double mydistance = 0;
1602            double otherdistance = 0;
1603
1604
1605            vector<NodePositionVector>::iterator posvecitr = posvec.begin();
1606            vector<NodePositionVector>::iterator posvecitr2 = posvec.begin();
1607
1608            while(posvecitr != posvec.end()){
1609                if((*posvecitr).nodeid == nodeId){
1610                    mydistance = CalculateDistanceBetweenPointsInMeters( (*posvecitr).nodepositionx ,
(*posvecitr).nodepositiony , sourcex , sourcey);
1611                    break;
1612                }
1613                ++posvecitr;
1614            }
1615
1616            while(posvecitr2 != posvec.end()){
1617                otherdistance = CalculateDistanceBetweenPointsInMeters((*posvecitr2).nodepositionx ,
(*posvecitr2).nodepositiony , sourcex , sourcey);
```

```
1618
1619                if(otherdistance < MAX_RANGE){
1620                    if(otherdistance > mydistance){
1621
1622                        //cout << "X - SourceX = " << (*posvecitr2).nodepositionx - sourcex << "Y - SourceY
= " << (*posvecitr2).nodepositiony - sourcey << endl;
1623
1624                        if(abs((*posvecitr2).nodepositionx - sourcex) < ROAD_WIDTH ||
abs((*posvecitr2).nodepositiony - sourcey < ROAD_WIDTH)){
1625                            //cout << "Node X = " <<(*posvecitr2).nodepositionx << "Source X = " << sourcex
<< "Node Y = " << (*posvecitr2).nodepositiony << "Source Y = " << sourcey << endl;
1626                            priorityнum++;
1627                        }
1628                    }
1629                    else if(otherdistance == mydistance && (*posvecitr2).nodeid > nodeId){
1630                        if(abs((*posvecitr2).nodepositionx - sourcex) < ROAD_WIDTH ||
abs((*posvecitr2).nodepositiony - sourcey < ROAD_WIDTH)){
1631                            priorityнum++;
1632                        }
1633                    }
1634                }
1635                ++posvecitr2;
1636            }
1637
1638            if(priorityнum > MAX_PRIORITY){
1639                //cout << "return priority = " << MAX_PRIORITY << endl;
1640                return MAX_PRIORITY;
1641            }
1642            //cout << "return priority = " << priorityнum << endl;
1643            return priorityнum;
1644        }
1645
1811
1812    inline
1813        void FloodingApplication::ScheduleRebroadcastEvent(
1814        const FloodingPayloadType& floodingPayload ,
```

```
1815        const unsigned int packetPayloadSizeBytes)
1816    {
1817        const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
1818        const TimeType minWaitingPeriod = floodingPayload.minWaitingPeriod;
1819        const TimeType maxWaitingPeriod = floodingPayload.maxWaitingPeriod;
1820
1821        TimeType nextPacketTime = currentTime + minWaitingPeriod;
1822
1823        int mypriority = 0;
1824
1825        //比較手法=ON
1826        //cout << "[before1]nextPacketTime = " << nextPacketTime <<  endl;
1827        /*
1828        nextPacketTime += static_cast<TimeType>(
1829            aRandomNumberGeneratorPtr->GenerateRandomDouble() * (maxWaitingPeriod - minWaitingPeriod));
1830            */
1831            /*
1832
1833        //cout << "[after1]nextPacketTime = " << nextPacketTime <<  endl;
1834
1835        //比較手法 2=ON
1836        /*
1837        mypriority = PrioritySolver2(nodeId , floodingPayload.nodePositionX ,
    floodingPayload.nodePositionY);
1838        nextPacketTime += static_cast<TimeType>(((double)maxWaitingPeriod - (double)minWaitingPeriod) /
    (double)SPLIT_NUM * (double)mypriority);
1839        */
1840            /*
1841
1842        //提案手法=ON
1843        //nextPacketTime += minWaitingPeriod;
1844
1845        //優先度決定関数
1846        //提案手法=ON
1847        //mypriority = PrioritySolver(nodeId , floodingPayload.nodePositionX ,
    floodingPayload.nodePositionY);
```

```
1848
1849            //cout << "[before2]nextPacketTime = " << nextPacketTime <<  endl;
1850            //cout << "maxWaitingPeriod - minWaitingPeriod = " << maxWaitingPeriod - minWaitingPeriod <<  endl;
1851            //cout << "((double)maxWaitingPeriod - (double)minWaitingPeriod) / (double)MAX_PRIORITY = " <<
    ((double)maxWaitingPeriod - (double)minWaitingPeriod) / (double)MAX_PRIORITY <<  endl;
1852
1853
1854            //nextPacketTime += static_cast<TimeType>(((double)maxWaitingPeriod - (double)minWaitingPeriod) /
    (double)MAX_PRIORITY * (double)mypriority);
1855
1856            //cout << "[after2]nextPacketTime = " << nextPacketTime <<  endl;
1857
1858            simulationEngineInterfacePtr->ScheduleEvent(
1859                unique_ptr<SimulationEvent>(
1860                new FloodingRebroadcastEvent(
1861                shared_from_this() ,
1862                floodingPayload ,
1863                packetPayloadSizeBytes)) ,
1864                nextPacketTime);
1865
1866        }//ScheduleRebroadcastEvent//
1867        */
1868
1869
1870        inline
1871            void FloodingApplication::ReceivePacket(unique_ptr<Packet>& packetPtr)
1872        {
1873            FloodingPayloadType floodingPayload =
1874                packetPtr->GetAndReinterpretPayloadData<FloodingPayloadType>();
1875
1876            assert(floodingPayload.maxHopCount != 0);
1877            floodingPayload.maxHopCount--;
1878
1879            IncrementCountOfPacketReceived(floodingPayload.id);
1880            UpdateMinDistanceBetweenNodesInMeters(floodingPayload);
1881
```

```
1882          const TimeType delay =
1883              simulationEngineInterfacePtr->CurrentTime() - floodingPayload.broadcastTime;
1884
1885          //吉川変更
1886          const TimeType currentTime = simulationEngineInterfacePtr->CurrentTime();
1887
1888          ObjectMobilityPosition nodePosition;
1889          nodeMobilityModelPtr->GetPositionForTime(
1890              simulationEngineInterfacePtr->CurrentTime() , nodePosition);
1891
          double distanceInMeters = CalculateDistanceBetweenPointsInMeters(nodePosition.X_PositionMeters() ,
1892  nodePosition.Y_PositionMeters() , floodingPayload.sourceNodePositionX ,
      floodingPayload.sourceNodePositionY);
1893
1894          //統計部分は MAC 層に移植
1895          //cout << "[APPLayer]ReceivePacket Completed :: NODEID = " << nodeId << endl;
1896
1897          if (IsPacketReceived(floodingPayload)) {
1898
1899              OutputTraceAndStatsForReceivePacket(
1900                  string(floodingPayload.id.applicationIdChar) ,
1901                  floodingPayload.id.sequenceNumber ,
1902                  packetPtr->GetPacketId() ,
1903                  packetPtr->LengthBytes() ,
1904                  delay);
1905
1906              //cout << "[APPLayer]ScheduleRebroadcastEvent :: NODEID = " << nodeId << endl;
1907              ScheduleRebroadcastEvent(floodingPayload , packetPtr->LengthBytes());
1908          }
1909          else {
1910              OutputTraceAndStatsForDiscardPacket(
1911                  string(floodingPayload.id.applicationIdChar) ,
1912                  floodingPayload.id.sequenceNumber ,
1913                  packetPtr->GetPacketId() ,
1914                  packetPtr->LengthBytes() ,
1915                  delay);
```

```
1916          }//if//
1917
1918          DeleteDummyUniquePtrIfItIsDummy(packetPtr);
1919          packetPtr = nullptr;
1920
1921      }//ReceivePacket//
```

Dot11_mac.h

```
2335
2336  inline
2337  void Dot11Mac::StartPacketSendProcessForAnAccessCategory(
2338      const unsigned int accessCategoryIndex ,
2339      const bool forceBackoff)
2340  {
2341      //吉川追加変数
2342      double nodePositionX = 0;
2343      double nodePositionY = 0;
2344      int mypriority = 0;
2345      unsigned int sourcenodeid = 0;
2346      unsigned int sequencenumber = 0;
2347
2348
2349      EdcaAccessCategoryInfo& accessCategoryInfo = accessCategories[accessCategoryIndex];
2350
2351      accessCategoryInfo.tryingToJumpOnMediumWithoutABackoff = false;
2352
2353      if ((accessCategoryInfo.currentPacketPtr == nullptr) &&
2354          (accessCategoryInfo.currentAggregateFramePtr == nullptr)) {
2355
2356          accessCategoryInfo.hasPacketToSend =
2357              ThereAreQueuedPacketsForAccessCategory(accessCategoryIndex);
2358
2359      }//if//
2360
2361      if ((!forceBackoff) && (!accessCategoryInfo.hasPacketToSend)) {
2362          accessCategoryInfo.currentNonExtendedBackoffDuration = INFINITE_TIME;
```

```
2363
2364            return;
2365
2366        }//if//
2367
2368        //情報の取得と格納と変換
2369        GetPacketPtr(accessCategoryIndex);
2370        if(accessCategoryInfo.currentPacketPtr != nullptr){
2371            unsigned char protocolNum;
2372
            ScenSim::IpHeaderOverlayModel
2373    ipHeader(accessCategoryInfo.currentPacketPtr->GetRawPayloadData() ,
        accessCategoryInfo.currentPacketPtr->LengthBytes());
2374
2375            unsigned int ipHeaderLength;
2376            ipHeader.GetHeaderTotalLengthAndNextHeaderProtocolCode(ipHeaderLength , protocolNum);
2377
2378            ipHeader.StopOverlayingHeader();
2379
            ScenSim::UdpHeaderType udpHeader =
2380  accessCategoryInfo.currentPacketPtr->GetAndReinterpretPayloadData<ScenSim::UdpHeaderType>();
2381
2382            ScenSim::FloodingApplication::FloodingPayloadType floodingPayload =

2383  accessCategoryInfo.currentPacketPtr->GetAndReinterpretPayloadData<ScenSim::FloodingApplication::
        FloodingPayloadType>((sizeof(ScenSim::UdpHeaderType)) + ipHeaderLength);
2384
2385            nodePositionX = floodingPayload.nodePositionX;
2386            nodePositionY = floodingPayload.nodePositionY;
2387            //ACK のために MAC 層で priority を実装する
2388            //mypriority = floodingPayload.mypriority;
2389            sourcenodeid = floodingPayload.id.sourceNodeId;
2390            sequencenumber = floodingPayload.id.sequenceNumber;
2391
2392            //APP 層で計算した myPriority を使用する
2393            mypriority = floodingPayload.mypriority;
```

```
2394          //mypriority = MacPrioritySolver(floodingPayload.myRelayNodeId  ,  nodeId);
2395
2396          //cout << "[MACLayer]Read Payload :: Source + SeqNum = " << sourcenodeid << " + " <<
sequencenumber << " :: NodeId = " << nodeId << endl;
2397
2398          ReturnPacketPtr(accessCategoryIndex);
2399      }
2400
2401      switch (macState) {
2402      case BusyMediumState:
2403      case CtsOrAckTransmissionState:
2404      case WaitingForNavExpirationState:
2405      case WaitingForCtsState:
2406      case WaitingForAckState:
2407      case ChangingChannelsState:
2408      {
2409
2410          //提案手法=ON
2411          accessCategoryInfo.currentNumOfBackoffSlots = mypriority;
2412
2413          accessCategoryInfo.currentNonExtendedBackoffDuration =
2414              CalculateNonExtendedBackoffDuration(accessCategoryInfo);
2415
2416          break;
2417
2418          /*旧実装
2419          accessCategoryInfo.currentNumOfBackoffSlots =
2420              aRandomNumberGenerator.GenerateRandomInt(0 ,
accessCategoryInfo.currentContentionWindowSlots);
2421
2422          accessCategoryInfo.currentNonExtendedBackoffDuration =
2423              CalculateNonExtendedBackoffDuration(accessCategoryInfo);
2424
2425          break;
2426          */
2427      }
```

```
2428     case IdleState:
2429     case WaitingForIfsAndBackoffState:
2430     {
2431         //cout << "[MACLayer-DEBUG] CALL STARTPACKETSENDPROCESS 1 :: NodeId = " << nodeId << endl;
2432         if (disabledToJumpOnMediumWithoutBackoff) {
2433 
2434             //cout << "[MACLayer-DEBUG] CALL STARTPACKETSENDPROCESS 2 :: NodeId = " << nodeId << endl;
2435 
2436             //提案手法
2437             accessCategoryInfo.currentNumOfBackoffSlots = mypriority;
2438 
2439             /*
2440             accessCategoryInfo.currentNumOfBackoffSlots =
2441                 aRandomNumberGenerator.GenerateRandomInt(0 , accessCategoryInfo.currentContentionWindowSlots);
2442             // Note: Add complete IFS even if channel is completely idle.
2443             */
2444 
2445             accessCategoryInfo.currentNonExtendedBackoffDuration =
2446                 CalculateNonExtendedBackoffDuration(accessCategoryInfo);
2447 
2448             accessCategoryInfo.tryingToJumpOnMediumWithoutABackoff = false;
2449         }
2450         else {
2451 
2452             //cout << "[MACLayer-DEBUG] CALL STARTPACKETSENDPROCESS 3 :: NodeId = " << nodeId << endl;
2453 
2454             accessCategoryInfo.currentNumOfBackoffSlots = 0;
2455             accessCategoryInfo.currentNonExtendedBackoffDuration =
2456                 CalculateNonExtendedDurationForJumpingOnMediumWithNoBackoff(accessCategoryInfo);
2457             accessCategoryInfo.tryingToJumpOnMediumWithoutABackoff = true;
2458         }
2459 
2460         (*this).StartBackoffForCategory(accessCategoryIndex);
2461 
2462         break;
```

```
2463
2464        }
2465        default:
2466            assert(false); abort(); break;
2467        }//switch//
2468
2469    }//StartPacketSendProcessForAnAccessCategory//
2470
6180    inline
6181    void Dot11Mac::ReceiveFrameFromPhy(
6182        const Packet& aFrame ,
6183        const TransmissionParametersType& receivedFrameTxParameters)
6184    {
6185        using std::max;
6186
6187        (*this).lastFrameReceivedWasCorrupt = false;
6188
6189        const CommonFrameHeaderType& header =
    aFrame.GetAndReinterpretPayloadData<CommonFrameHeaderType>();
6190
6191        if (FrameIsForThisNode(aFrame)) {
6192
6193            OutputTraceAndStatsForFrameReceive(aFrame);
6194
6195            switch (header.frameControlField.frameTypeAndSubtype) {
6196            case RTS_FRAME_TYPE_CODE: {
6197                const RequestToSendFrameType& rtsFrame =
    aFrame.GetAndReinterpretPayloadData<RequestToSendFrameType>();
6198
6199                (*this).ProcessRequestToSendFrame(rtsFrame , receivedFrameTxParameters);
6200
6201                break;
6202            }
6203            case CTS_FRAME_TYPE_CODE: {
6204                const ClearToSendFrameType& ctsFrame =
    aFrame.GetAndReinterpretPayloadData<ClearToSendFrameType>();
```

```
6205
6206            (*this).ProcessClearToSendFrame(ctsFrame);
6207
6208            break;
6209        }
6210        case NULL_FRAME_TYPE_CODE:
6211            (*this).ProcessNullFrame(aFrame , receivedFrameTxParameters);
6212
6213            break;
6214        case QOS_DATA_FRAME_TYPE_CODE: {
6215
6216            //データフレームはここにくる
6217            //吉川追加
6218            unsigned char protocolNum;
6219
6220            ScenSim::IpHeaderOverlayModel ipHeader(aFrame.GetRawPayloadData() , aFrame.LengthBytes());
6221
6222            unsigned int ipHeaderLength;
6223            ipHeader.GetHeaderTotalLengthAndNextHeaderProtocolCode(ipHeaderLength , protocolNum);
6224
6225            ScenSim::FloodingApplication::FloodingPayloadType floodingPayload =
6226 aFrame.GetAndReinterpretPayloadData<ScenSim::FloodingApplication::FloodingPayloadType>((sizeof(ScenSim::UdpHeaderType)) + ipHeaderLength + sizeof(QosDataFrameHeaderType));
6227
6228            //cout << "[MACLayer]ReceiveFrame :: Source + Seq = " << floodingPayload.id.sourceNodeId << " + " << floodingPayload.id.sequenceNumber << " :: NODEID = "<< nodeId << endl;
6229
6230            IncrementCountOfPacketReceived(floodingPayload.id);
6231
6232            int check = 0; //重複チェック
6233            double delay_current = 0;
6234            int temp = 0;
6235
6236            if(IsPacketFirstlyReceived(floodingPayload.id)){
```

```
6237                vector<StatisticVector>::iterator statvecitr = statvec.begin();
6238                while(statvecitr != statvec.end()){
6239                    if((*statvecitr).sourcenodeid == floodingPayload.id.sourceNodeId &&
(*statvecitr).sequencenumber == floodingPayload.id.sequenceNumber){
6240                        (*statvecitr).receivecount++;
6241                        //平均 delay と最大と最小
6242                        if((*statvecitr).endtoenddelay_min >
(simEngineInterfacePtr->CurrentTime() - (*statvecitr).generatetime)){
6243                            (*statvecitr).endtoenddelay_min =
(simEngineInterfacePtr->CurrentTime() - (*statvecitr).generatetime);
6244                        }
6245                        if((*statvecitr).endtoenddelay_max <
(simEngineInterfacePtr->CurrentTime() - (*statvecitr).generatetime)){
6246                            (*statvecitr).endtoenddelay_max =
(simEngineInterfacePtr->CurrentTime() - (*statvecitr).generatetime);
6247                        }
6248
6249                        (*statvecitr).endtoenddelay = (*statvecitr).endtoenddelay +
((simEngineInterfacePtr->CurrentTime() - (*statvecitr).generatetime) / MILLI_SECOND);
6250
6251                        delay_current = simEngineInterfacePtr->CurrentTime() -
(*statvecitr).generatetime;
6252                        temp = (int)( delay_current / ( 100 * MILLI_SECOND));
6253                        if(temp < 11){
6254                            delay_distribution[temp] = delay_distribution[temp] + 1;
6255                        }
6256                        else{
6257                            delay_distribution[10] = delay_distribution[10] + 1;
6258                        }
6259                        break;
6260                    }
6261                    ++statvecitr;
6262                }
6263            }
6264
6265            //統計用
```

```
6266            total_success_phy++;
6267
6268            (*this).ProcessDataFrame(aFrame , receivedFrameTxParameters);
6269
6270            break;
6271        }
6272        case ACK_FRAME_TYPE_CODE: {
6273            const AcknowledgementAkaAckFrameType& ackFrame =
6274                aFrame.GetAndReinterpretPayloadData<AcknowledgementAkaAckFrameType>();
6275
6276            (*this).ProcessAcknowledgementFrame(ackFrame);
6277
6278            break;
6279        }
6280        case BLOCK_ACK_REQUEST_FRAME_TYPE_CODE: {
6281
6282            const BlockAcknowledgementRequestFrameType& blockAckRequestFrame =
6283                aFrame.GetAndReinterpretPayloadData<BlockAcknowledgementRequestFrameType>();
6284
6285            (*this).ProcessBlockAckRequestFrame(blockAckRequestFrame ,
    receivedFrameTxParameters);
6286
6287            break;
6288        }
6289        case BLOCK_ACK_FRAME_TYPE_CODE: {
6290
6291            const BlockAcknowledgementFrameType& blockAckFrame =
6292                aFrame.GetAndReinterpretPayloadData<BlockAcknowledgementFrameType>();
6293
6294            (*this).ProcessBlockAckFrame(blockAckFrame);
6295
6296            break;
6297        }
6298        case POWER_SAVE_POLL_FRAME_TYPE_CODE: {
6299            const PowerSavePollFrameType& psPollFrame =
6300                aFrame.GetAndReinterpretPayloadData<PowerSavePollFrameType>();
```

```
6301
6302            (*this).ProcessPowerSavePollFrame(psPollFrame , receivedFrameTxParameters);
6303
6304            break;
6305        }
6306        default:
6307            if (IsAManagementFrameTypeCode(header.frameControlField.frameTypeAndSubtype)) {
6308                (*this).ProcessManagementFrame(aFrame , receivedFrameTxParameters);
6309            } else {
6310                assert(false); abort();
6311            }
6312            break;
6313
6314        }//switch//
6315
6316    }
6317    else {
6318        // Not for this node.
6319
6320        // Set Virtual(Software) Carrier Sense aka NAV.
6321
6322        TimeType endOfDurationTime;
6323
6324        if (header.frameControlField.frameTypeAndSubtype != POWER_SAVE_POLL_FRAME_TYPE_CODE) {
6325            endOfDurationTime =
6326                simEngineInterfacePtr->CurrentTime() + (header.duration * MICRO_SECOND);
6327        }
6328        else {
6329            // Duration field is not a duration but an Assocation ID.
6330            endOfDurationTime =
6331                simEngineInterfacePtr->CurrentTime() +
6332                CalculateNavDurationForAckedDataFrame(receivedFrameTxParameters);
6333        }//if//
6334
6335        mediumReservedUntilTimeAkaNAV = max(mediumReservedUntilTimeAkaNAV , endOfDurationTime);
6336
```

```
6337          bool wasInWaitingForCtsOrAckState =
6338              ((macState == WaitingForCtsState) || (macState == WaitingForAckState));
6339
6340          macState = BusyMediumState;
6341
6342          if (wasInWaitingForCtsOrAckState) {
6343              // Received a packet but wasn't the desired ACK or CTS for this node ,  must retry.
6344              (*this).NeverReceivedClearToSendOrAcknowledgementAction();
6345
6346          }//if//
6347      }//if//
6348
6349  }//ReceiveFrameFromPhy//
```

Scensim_application.cpp

```
  21
  22  int totalgetcounter = 0;
  23  int totalcounter = 0;
  24
  25  int total_error_phy = 0;
  26  int total_success_phy = 0;
  27  int error_noise_phy = 0;
  28  int error_duringtransmission_phy = 0;
  29  int error_tooweak_phy = 0;
  30  int error_signalscaptured_phy = 0;
  31  int total_send_mac = 0;
  32  double total_busy_time_ms = 0;
  33  double total_idle_time_ms = 0;
  34
  35  int cancel_on_mac = 0;
  36
  37  int interfering_count = 0;
  38  int weaksig_count = 0;
  39  int signalduring_count = 0;
  40  int CorruptedFrame = 0;
  41
```

```
  42 int diff_distribution[11] = { 0 };
  43 int delay_distribution[11] = { 0 };
  44 int generate_infos = 0;
  45 int RebroadCastTimes = 0;
  46
  47 int SignalCaptureTimes = 0;
  48 int FramesWithErros = 0;
  49 int FramesTransmitted = 0;
  50 int FramesReceived = 0;
  51
  52 int Intersection_Priority = 1;
  53
  54 vector<StatisticVector> statvec;
  55 vector<StatisticVector> rebroadcast_count;
  56 vector<NodePositionVector> posvec;
  57
  58
1051 //---------------------------------------------------------------------------
1052
1053 void ApplicationMaker::ReadFloodingFromConfig(
1054     const ParameterDatabaseReader& theParameterDatabaseReader ,
1055     const ApplicationInstanceInfo& applicationInstanceId)
1056 {
1057     const NodeIdType& sourceNodeId = applicationInstanceId.sourceNodeId;
1058     const InterfaceOrInstanceIdType& instanceId = applicationInstanceId.instanceId;
1059
1060     const NodeIdType destinationNodeIdOrAnyNodeId =
1061         App_ConvertStringToNodeIdOrAnyNodeId(
1062             theParameterDatabaseReader.ReadString(
1063                 "flooding-destination" , sourceNodeId , instanceId));
1064
1065     const unsigned short defaultDestinationPortId =
1066         applicationInstanceId.GetDefaultDestinationPortNumber();
1067
1068     assert(destinationNodeIdOrAnyNodeId == ANY_NODEID);
1069
```

```
1070     if (floodingApplicationPtr == nullptr) {
1071         shared_ptr<FloodingApplication> appPtr(
1072             new FloodingApplication(
1073                 theParameterDatabaseReader ,
1074                 simulationEngineInterfacePtr ,
1075                 instanceId ,
1076                 nodeId ,
1077                 defaultDestinationPortId));
1078 
1079         appLayerPtr->AddApp(appPtr);
1080         appPtr->CompleteInitialization();
1081         floodingApplicationPtr = appPtr;
1082     }//if//
1083 
1084     if (sourceNodeId == nodeId) {
1085         floodingApplicationPtr->AddSenderSetting(
1086             theParameterDatabaseReader ,
1087             instanceId);
1088 
1089         if(nodeId == 1){
1090             //全体統計用関数設定
1091             //統計出力用
1092             cout << "[SETTINGS] NODES =  " << NODES << "MAX_PRIORITY = " << MAX_PRIORITY << endl;
1093             cout << "time , getcount , totalcount , total_diff , delay , err_rate , success , error , cancel_on_mac , total_send , busytime , idletime , interfering , weaksignal , signalduring" << endl;
1094             floodingApplicationPtr->initStatistic();
1095             statvec.reserve(50000);
1096         }
1097 
1098         //吉川追加 統計用関数設定
1099         floodingApplicationPtr->StatisticSetting();
1100 
1101 
1102     }//if//
1103 
1104     floodingApplicationPtr->AddReceiverSetting(instanceId);
```

```
1105
1106 }//ReadFloodingFromConfig//
```

付録C.　シナリオファイル

```
#Scenargie 1.7 VisualLab r15247

gui-portnumber-sim = 5000
gui-portnumber-pausecommand = 5001

visuallab-install-directory = C:/Scenargie
visuallab-case-name = 1600m4x4

#Instance general
#Component Simulation
seed = 123
simulation-time = 120.000000000
simulation-base-time = 2000-01-01T00:00:00
time-step-event-synchronization-step = 0.100000000
sim-postype = Cartesian
executable-name = C:/Scenargie/bin/sim.exe
trace-analyzer-lib = C:/Scenargie/bin/traceanalyzer.dll
trace-output-mode = Binary
trace-index-output = true
trace-output-file = 1600m4x4.trace
statistics-output-file = 1600m4x4.stat
statistics-output-for-no-data = true
network-static-route-file = 1600m4x4.routes
network-terminate-sim-when-routing-fails = false
trace-file-for-playback = 1600m4x4.trace
mobility-file-for-playback = 1600m4x4.mob.trace
statistics-file-for-playback = 1600m4x4.stat
progress-sim-time-output-interval-percents = 0.000000000
enable-unused-parameter-warnings = false

#Component GIS
gis-road-driving-side = left
gis-los-break-down-cureved-road-into-straight-roads = false
```

```
gis-number-entrances-to-building = 1
gis-number-entrances-to-station = 1
gis-number-entrances-to-busstop = 1
gis-number-entrances-to-park = 1
gis-road-set-intersection-margin = true
gis-signal-pattern-definition-file = SignalPattern.txt

#Component Antenna/Propagation
number-data-parallel-threads-for-propagation = 1
custom-antenna-file = default.ant
antenna-pattern-two-2d-to-3d-interpolation-algorithm-number = 1
antenna-patterns-are-in-legacy-format = false
material-file = default.material
moving-object-shape-file = default.oshp

#Component Dot11
dot11-bit-error-rate-curve-file = dot11modes.ber

#Component MultiAgent
multiagent-profile-file = 1600m4x4_AgentProfiles.txt
multiagent-behavior-file = 1600m4x4_AgentBehaviors.txt
gis-public-vehicle-file = 1600m4x4_VehicleTimeTable.txt
multiagent-start-time = 0.000000000
number-data-parallel-threads-for-multiagent = 1
multiagent-navigation-system-update-interval = 60.000000000


[101000001-101000025] is-member-of = IntersectionObjectType
[100000001-100000040] is-member-of = RoadObjectType
[102000001-102000016] is-member-of = BuildingObjectType
[1-500] is-member-of = Dot11pObjectType
#Instance general

#Instance 5.9GHzBand
#Component Channel
[5.9GHzBand] channel-frequency-mhz = 5900.000000000
```

```
[5.9GHzBand] channel-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-count = 7
[5.9GHzBand] channel-0-frequency-mhz = 5860.000000000
[5.9GHzBand] channel-0-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-1-frequency-mhz = 5870.000000000
[5.9GHzBand] channel-1-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-2-frequency-mhz = 5880.000000000
[5.9GHzBand] channel-2-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-3-frequency-mhz = 5890.000000000
[5.9GHzBand] channel-3-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-4-frequency-mhz = 5900.000000000
[5.9GHzBand] channel-4-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-5-frequency-mhz = 5910.000000000
[5.9GHzBand] channel-5-bandwidth-mhz = 10.000000000
[5.9GHzBand] channel-6-frequency-mhz = 5920.000000000
[5.9GHzBand] channel-6-bandwidth-mhz = 10.000000000
[5.9GHzBand] propagation-enable-mask-calculated-channel-interference = false
[5.9GHzBand] propagation-model = ITU-R_P.1411
[5.9GHzBand] p1411-los-calculation-policy = median
[5.9GHzBand] p1411-max-diffraction-count = 1
[5.9GHzBand] p1411-los-angle-degrees-between-roads = 1.000000000
[5.9GHzBand] p1411-nlos-max-distance-meters = 100000.000000000
[5.9GHzBand] p1411-enable-building-based-los-calculation = false
[5.9GHzBand] p1411-nlos1-calculation-policy = urban
[5.9GHzBand] p1411-nlos2-calculation-policy = urban
[5.9GHzBand] p1411-nlos2-loss-direction = Directional
[5.9GHzBand] p1411-nlos2-use-policy = AlwaysUse800To2000MHzCalculation
[5.9GHzBand] p1411-nlos2-extension = no
[5.9GHzBand] p1411-nlos2-use-larger-loss-at-nlos-bound = false
[5.9GHzBand] p1411-nlos800to2000-calculation-policy = lower
[5.9GHzBand] p1411-shf-effective-road-height-meters = 0.000000000
[5.9GHzBand] p1411-shf-short-distance-meters = 20.000000000
[5.9GHzBand] p1411-enable-propagation-between-terminals-located-below-rooftop-height-at-uhf =
false
[5.9GHzBand] p1411-building-height-differ-threshold-meters = 1.000000000
[5.9GHzBand] p1411-below-rooftop-calculation-policy = urban
```

```
105 [5.9GHzBand] p1411-well-below-rooftop-height-meters = 3.000000000
106 [5.9GHzBand] p1411-below-rooftop-location-percentage = 50
107 [5.9GHzBand] p1411-below-rooftop-transition-region-meters = 20.000000000
108 [5.9GHzBand] enable-propagation-delay = true
109 [5.9GHzBand] max-signal-propagation-meters = 100000000.000000000
110 [5.9GHzBand] propagation-allow-multiple-interfaces-on-same-channel = true
111 [5.9GHzBand] fading-model = OFF
112
113
114 #Instance Flooding496
115
116 #Instance Flooding497
117
118 #Instance general
119 #Component Common
120
121 #Component Position
122
123 #Component CommunicationObject
124
125 #Component SimulationObject
126 [1-500,100000001-100000040,101000001-101000025,102000001-102000016] trace-start-time =
    0.000000000
127
128 #Component GisObject
129 [100000001-100000040,101000001-101000025,102000001-102000016] gisobject-disable-time = inf_time
130 [100000001-100000040,101000001-101000025,102000001-102000016] gisobject-enable-time = inf_time
131 [100000001-100000040,101000001-101000025,102000001-102000016]
    gisobject-elevation-reference-type = GroundLevel
132
133 #Component Building
134 [102000001-102000016] commonbuilding-capacity = 100
135 [102000001-102000016] commonbuilding-roof-material = structure_default
136 [102000001-102000016] commonbuilding-wall-material = structure_default
137 [102000001-102000016] commonbuilding-floor-material = structure_default
138
```

```
#Component Road
[100000001-100000040] commonroad-capacity = 7.000000000
[100000001-100000040] commonroad-type = Road

#Component PointObject

#Component Intersection

#Component Network (Node)
[1-500] network-hop-limit = 64
[1-500] network-loopback-delay = 0.000000001

#Component Transport
[1-500] tcp-settings = Default

#Component Mobility
[1-500] mobility-model = STATIONARY
[1-500] mobility-granularity-meters = 1.000000000
[1-500] mobility-init-positions-file = 1600m4x4.pos
[1-500] mobility-contains-ground-height = false


#Instance Flooding498

#Instance Flooding499

#Instance Flooding500

#Instance dot11p
#Component Antenna/Propagation (Interface)
[1-500;dot11p] channel-model = 5.9GHzBand
[1-500;dot11p] antenna-model = OMNIDIRECTIONAL
[1-500;dot11p] max-antenna-gain-dbi = 0.000000000
[1-500;dot11p] antenna-height-meters = 1.500000000
[1-500;dot11p] antenna-azimuth-degrees = 0.000000000
[1-500;dot11p] antenna-elevation-degrees = 0.000000000
```

```
175 [1-500;dot11p] antenna-offset-meters = 0.000000000
176 [1-500;dot11p] antenna-offset-degreess = 0.000000000
177
178 #Component Routing
179
180 #Component Network (Interface)
181 [1-500;dot11p] network-address = 90.3.0.0 + $n
182 [1-500;dot11p] network-prefix-length-bits = 16
183 [1-500;dot11p] network-subnet-is-multihop = false
184 [1-500;dot11p] network-address-is-primary = false
185 [1-500;dot11p] network-allow-routing-back-out-same-interface = true
186 [1-500;dot11p] network-ignore-unregistered-protocol = false
187 [1-500;dot11p] network-mtu-bytes = 1500
188 [1-500;dot11p] mac-protocol = DOT11
189 [1-500;dot11p] interface-output-queue-max-packets-per-subq = 1000
190 [1-500;dot11p] interface-output-queue-max-bytes-per-subq = 1000000
191 [1-500;dot11p] network-enable-dhcp-client = false
192 [1-500;dot11p] network-enable-dhcp-server = false
193 [1-500;dot11p] network-dhcp-model = abstract
194 [1-500;dot11p] network-enable-ndp = false
195 [1-500;dot11p] network-enable-arp = false
196
197 #Component Dot11Mac
198 [1-500;dot11p] dot11-node-type = AD-HOC
199 [1-500;dot11p] dot11-initial-channel-number = 0
200 [1-500;dot11p] dot11-enable-high-throughput-mode = false
201 [1-500;dot11p] dot11-adaptive-rate-control-type = Static
202 [1-500;dot11p] dot11-modulation-and-coding = BPSK_0.5
203 [1-500;dot11p] dot11-modulation-and-coding-for-management-frames = BPSK_0.5
204 [1-500;dot11p] dot11-modulation-and-coding-for-broadcast = BPSK_0.5
205 [1-500;dot11p] dot11-ack-datarate-selection-type = SameAsData
206 [1-500;dot11p] dot11-rts-threshold-size-bytes = 2346
207 [1-500;dot11p] dot11-short-frame-retry-limit = 7
208 [1-500;dot11p] dot11-long-frame-retry-limit = 4
209 [1-500;dot11p] dot11-msdu-aggregation-is-enabled = false
210 [1-500;dot11p] dot11-max-aggregate-mpdu-size-bytes = 0
```

```
211 [1-500;dot11p] dot11-contention-window-min-slots = 15
212 [1-500;dot11p] dot11-contention-window-max-slots = 1023
213 [1-500;dot11p] dot11-disabled-to-jump-on-medium-without-backoff = false
214 [1-500;dot11p] dot11-qos-type = EDCA
215 [1-500;dot11p] dot11-max-packet-priority = 7
216 [1-500;dot11p] dot11-edca-category-0-priority-list = 1 2
217 [1-500;dot11p] dot11-edca-category-0-num-aifs-slots = 9
218 [1-500;dot11p] dot11-edca-category-0-contention-window-min-slots = 15
219 [1-500;dot11p] dot11-edca-category-0-contention-window-max-slots = 1023
220 [1-500;dot11p] dot11-edca-category-0-frame-lifetime = inf_time
221 [1-500;dot11p] dot11-edca-category-0-downlink-txop-duration = 0.000000000
222 [1-500;dot11p] dot11-edca-category-1-priority-list = 0 3
223 [1-500;dot11p] dot11-edca-category-1-num-aifs-slots = 6
224 [1-500;dot11p] dot11-edca-category-1-contention-window-min-slots = 15
225 [1-500;dot11p] dot11-edca-category-1-contention-window-max-slots = 1023
226 [1-500;dot11p] dot11-edca-category-1-frame-lifetime = inf_time
227 [1-500;dot11p] dot11-edca-category-1-downlink-txop-duration = 0.000000000
228 [1-500;dot11p] dot11-edca-category-2-priority-list = 4 5
229 [1-500;dot11p] dot11-edca-category-2-num-aifs-slots = 3
230 [1-500;dot11p] dot11-edca-category-2-contention-window-min-slots = 7
231 [1-500;dot11p] dot11-edca-category-2-contention-window-max-slots = 15
232 [1-500;dot11p] dot11-edca-category-2-frame-lifetime = inf_time
233 [1-500;dot11p] dot11-edca-category-2-downlink-txop-duration = 0.000000000
234 [1-500;dot11p] dot11-edca-category-3-priority-list = 6 7
235 [1-500;dot11p] dot11-edca-category-3-num-aifs-slots = 2
236 [1-500;dot11p] dot11-edca-category-3-contention-window-min-slots = 3
237 [1-500;dot11p] dot11-edca-category-3-contention-window-max-slots = 7
238 [1-500;dot11p] dot11-edca-category-3-frame-lifetime = inf_time
239 [1-500;dot11p] dot11-edca-category-3-downlink-txop-duration = 0.000000000
240
241 #Component Dot11Phy
242 [1-500;dot11p] dot11-phy-protocol = IEEE802.11
243 [1-500;dot11p] dot11-tx-power-specified-by = PhyLayer
244 [1-500;dot11p] dot11-tx-power-dbm = 20.000000000
245 [1-500;dot11p] dot11-radio-noise-figure-db = 10.000000000
246 [1-500;dot11p] dot11-preamble-detection-power-threshold-dbm = -85.000000000
```

```
247 [1-500;dot11p] dot11-energy-detection-power-threshold-dbm = -65.000000000
248 [1-500;dot11p] dot11-signal-capture-ratio-threshold-db = 10.000000000
249 [1-500;dot11p] dot11-ofdm-symbol-duration = 0.000008000
250 [1-500;dot11p] dot11-slot-time = 0.000013000
251 [1-500;dot11p] dot11-sifs-time = 0.000032000
252 [1-500;dot11p] dot11-rx-tx-turnaround-time = 0.000002000
253 [1-500;dot11p] dot11-phy-rx-start-delay = 0.000049000
254 [1-500;dot11p] dot11-preamble-length-duration = 0.000032000
255 [1-500;dot11p] dot11-plcp-header-length-duration = 0.000008000
256 [1-500;dot11p] dot11-phy-high-throughput-header-additional-duration = 0.000008000
257 [1-500;dot11p] dot11-phy-high-throughput-header-additional-per-stream-duration = 0.000004000
258
259
1249
1250 gis-object-position-in-latlong-degree = false
1251 gis-object-file-path = shapes/
1252 gis-object-files = intersection.shp road.shp signal.shp busstop.shp building.shp
1253
1254 #Component Flooding
1255 [1;Flooding1] flooding-destination = *
1256 [2;Flooding2] flooding-destination = *
1257 [3;Flooding3] flooding-destination = *
    :
    以下省略
    :
1752 [498;Flooding498] flooding-destination = *
1753 [499;Flooding499] flooding-destination = *
1754 [500;Flooding500] flooding-destination = *
1755 [1;Flooding1] flooding-payload-size-bytes = 128
1756 [2;Flooding2] flooding-payload-size-bytes = 128
1757 [3;Flooding3] flooding-payload-size-bytes = 128
    :
    以下省略
    :
2252 [498;Flooding498] flooding-payload-size-bytes = 128
2253 [499;Flooding499] flooding-payload-size-bytes = 128
```

```
2254 [500;Flooding500] flooding-payload-size-bytes = 128
2255
2256 [1-500] flooding-interval = 8.000000000
2257
2258 [1;Flooding1] flooding-start-time = 10.000000000
2259 [2;Flooding2] flooding-start-time = 10.000000000
2260 [3;Flooding3] flooding-start-time = 10.000000000
     :
     以下省略
     :
2755 [498;Flooding498] flooding-start-time = 10.000000000
2756 [499;Flooding499] flooding-start-time = 10.000000000
2757 [500;Flooding500] flooding-start-time = 10.000000000
2758 [1;Flooding1] flooding-end-time = 110.000000000
2759 [2;Flooding2] flooding-end-time = 110.000000000
2760 [3;Flooding3] flooding-end-time = 110.000000000
     :
     以下省略
     :
3255 [498;Flooding498] flooding-end-time = 110.000000000
3256 [499;Flooding499] flooding-end-time = 110.000000000
3257 [500;Flooding500] flooding-end-time = 110.000000000
3258 [1;Flooding1] flooding-priority = 0
3259 [2;Flooding2] flooding-priority = 0
3260 [3;Flooding3] flooding-priority = 0
     :
     以下省略
     :
3755 [498;Flooding498] flooding-priority = 0
3756 [499;Flooding499] flooding-priority = 0
3757 [500;Flooding500] flooding-priority = 0
3758
3759 [1-500] flooding-max-hop-count = 100
3760
3761 [1;Flooding1] flooding-min-waiting-period = 0.100000000
3762 [2;Flooding2] flooding-min-waiting-period = 0.100000000
```

```
3763 [3;Flooding3] flooding-min-waiting-period = 0.100000000
     :
     以下省略
     :
4258 [498;Flooding498] flooding-min-waiting-period = 0.100000000
4259 [499;Flooding499] flooding-min-waiting-period = 0.100000000
4260 [500;Flooding500] flooding-min-waiting-period = 0.100000000
4261 [1;Flooding1] flooding-max-waiting-period = 0.500000000
4262 [2;Flooding2] flooding-max-waiting-period = 0.500000000
4263 [3;Flooding3] flooding-max-waiting-period = 0.500000000
     :
     以下省略
     :
4758 [498;Flooding498] flooding-max-waiting-period = 0.500000000
4759 [499;Flooding499] flooding-max-waiting-period = 0.500000000
4760 [500;Flooding500] flooding-max-waiting-period = 0.500000000
4761
4762 [1-500] flooding-counter-threshold = 100000
4763
4764 [1;Flooding1] flooding-distance-threshold-in-meters = 0.000000000
4765 [2;Flooding2] flooding-distance-threshold-in-meters = 0.000000000
4766 [3;Flooding3] flooding-distance-threshold-in-meters = 0.000000000
     :
     以下省略
     :
5261 [498;Flooding498] flooding-distance-threshold-in-meters = 0.000000000
5262 [499;Flooding499] flooding-distance-threshold-in-meters = 0.000000000
5263 [500;Flooding500] flooding-distance-threshold-in-meters = 0.000000000
5264 [1;Flooding1] flooding-start-time-max-jitter = 1.000000000
5265 [2;Flooding2] flooding-start-time-max-jitter = 1.000000000
5266 [3;Flooding3] flooding-start-time-max-jitter = 1.000000000
     :
     以下省略
     :
5761 [498;Flooding498] flooding-start-time-max-jitter = 1.000000000
5762 [499;Flooding499] flooding-start-time-max-jitter = 1.000000000
```

```
5763 [500;Flooding500] flooding-start-time-max-jitter = 1.000000000
5764 [1;Flooding1] flooding-use-virtual-payload = false
5765 [2;Flooding2] flooding-use-virtual-payload = false
5766 [3;Flooding3] flooding-use-virtual-payload = false
     :
     以下省略
     :
6261 [498;Flooding498] flooding-use-virtual-payload = false
6262 [499;Flooding499] flooding-use-virtual-payload = false
6263 [500;Flooding500] flooding-use-virtual-payload = false
6264 [1;Flooding1] flooding-auto-port-mode = true
6265 [2;Flooding2] flooding-auto-port-mode = true
6266 [3;Flooding3] flooding-auto-port-mode = true
     :
     以下省略
     :
6761 [498;Flooding498] flooding-auto-port-mode = true
6762 [499;Flooding499] flooding-auto-port-mode = true
6763 [500;Flooding500] flooding-auto-port-mode = true
6764
6765
6766 statistics-configuration-file = 1600m4x4.statconfig
```