無線通信における 環境情報を利用した上位層における通信制御手法の検討

飯尾　明日香†　　小口　正人†

† お茶の水女子大学　〒 112-8610 東京都文京区大塚 2-1-1
E-mail: †asuka-i@ogl.is.ocha.ac.jp, ††oguchi@computer.org

あらまし　現在，自動車には無数のセンサが搭載されており，それらのセンサからは，自動車の状況を特徴づけるような環境情報，すなわち，Context を取得できる．センサ情報から生成された Context は，主に車の事故防止やドライバの完全運転支援のため車両制御に利用されているが，これは無線通信における最適なパラメータ設定にも利用可能であると考えられる．無線通信において，一般に通信パラメータは周囲の状況に関わらず静的に設定されているため，端末は必ずしも効率の良い通信設定がされているとはいえないことから，改良の余地があると考えられる．

本研究では，通信の上位層である TCP の輻輳制御に着目し，ここに環境情報として周辺端末数を利用することで，通信性能を改善する手法を検討する．TCP の輻輳制御において性能を決定する重要なパラメータである輻輳ウィンドウサイズは，周辺の端末数に関わらず輻輳制御アルゴリズムにより制御されているため，各端末は必ずしも効率的な通信を行っているとは言えない．本稿では，輻輳制御を行う TCP と輻輳ウィンドウサイズを大きな値に保つ TCP を用いてシミュレーションを行い，それぞれの通信性能を比較した．

キーワード　環境情報，Context，TCP（Transmission Control Protocol），輻輳制御方式

A Study of the Communication Control in Wireless Networks by Using Environmental Information on Transport Layer.

Asuka IIO† and Masato OGUCHI†

† Ochanomizu University　　2-1-1 Otsuka, Bunkyou-ku, Tokyo, 112-8610, JAPAN
E-mail: †asuka-i@ogl.is.ocha.ac.jp, ††oguchi@computer.org

Key words　Environment information，Context，TCP（Transmission Control Protocol），Congestion Control Algorithm

1. はじめに

現在，自動車には数百種類のセンサが搭載されており，自動車はこれらのセンサから GPS（全地球測位システム）による車両位置を始め，車両間隔，スピード，周辺端末数，加速度など数多くの情報を取得できる．車載センサはドライバの安全機能強化の面から高精度化が進んでおり，搭載数も年々増加の傾向にある．このようなセンサから得た周囲の環境情報を Context と呼ぶが，自動車において言えば，この Context は事故防止・車両安定制御等に役立てられている．

一方，近年 Wireless Local Area Network（WLAN）による通信が一般化しているが，WLAN において通信に用いられる各パラメータは各々が独自のアルゴリズムに従って設定されており，その場の環境に応じた通信設定は行われない．したがって，センサから環境情報を取得可能な端末であっても資源を効率的に利用した通信が行われておらず，必ずしも最適ではない状態で通信を行っているという問題がある [1][2]．そこで，本研究では，環境情報を適切な無線通信パラメータ設定に利用し，各端末が周囲の状況に応じた設定を行うことで，通信効率を向上させる手法の検討を行う．

本稿では，環境情報の利用先として上位層の Transmission Control Protocol（TCP）に着目した．TCP において，確認応答（ACK）を受けずに送信できるパケットの最大数である輻輳ウィンドウ（cwnd）サイズは輻輳制御アルゴリズムにより制御されているため，各端末は必ずしも周辺の状況に合った通信を行っているとは言えない．そこで，輻輳制御アルゴリズムにより cwnd サイズが制御される一般的な TCP を用いて通信を行った場合に対し，本稿ではまず，cwnd サイズを一定値とする TCP を用いて通信を行った場合において，端末数や往復遅延時間（RTT）に応じたスループットの変化をシミュレーションにより

測定し，比較する．これにより，今後環境情報に基づき cwnd サイズの最適値を決定する制御を行うための基礎的な検討を行う．なお，実験にはネットワークシミュレータ OMNeT++ version4.1 を使用した．

本論文では移行の 2 章において Context について述べ，関連研究を紹介する．3 章では TCP の輻輳制御について述べ，また 4 章では OMNeT++を紹介する．5 章では検証実験の概要および実験結果，考察について述べ，6 章でまとめと今後の課題を述べる．

2. Context

2.1 Context とは

Context は「実体を特徴づけることのできるあらゆる情報」と定義されている．ここで，実体とはユーザとアプリケーションの間のやり取りに関連していると考えられる人，場所，物のことを指しており，これにはユーザとアプリケーション自体も含んでいる．Context をこのように定義することで，あるアプリケーションを開発する際に必要とする，実体のその時の状況がわかるような環境情報の列挙がより容易になる．[1][2]

図 1 は，センサ情報の取得から Context 生成の流れを表している．無数のセンサから取得した環境情報を集め，さまざまなアプリケーションで利用しやすいように抽象化することにより Context は生成され，データベースで管理されている．

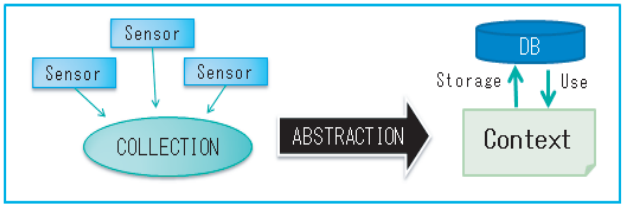

図 1　センサ情報から Context を生成

Context はさまざまなアプリケーションで利用することができることから，通信性能の改善にも利用可能であると考えられる．

2.2 Context-Aware Computing

ユーザに対し，関連する情報やサービスを提供するために Context を使用するシステムを Context-Aware であると呼ぶ．たとえ人が意識しなくとも，コンピュータがリアルタイムで実体の情報を収集・処理し，Context 化して利用する技術や概念のことを示している．

Context-Awere Computing は，Context を利用して企業や個人の情報利用環境を改善しようとする取り組みで，システムやユーザがその時に必要としている情報や機能を判断し，ユーザにサービスとして提供するものである．つまり，様々なシステムやセンサなどから得られる情報，すなわち，Context と人とをリアルタイムで結びつけ，ユーザがその時に必要な情報や機能を判断して，サービスとして提供する技術である．これは多数の技術の連携によって実現するが，近年，システム連携の仕組みが発展してきたことから Context-Awere Computing は実現が期待され，重要性が高まっている．

2.3 関 連 研 究

Context 利用に関連する研究として参考文献 [3] に示した Context-aware Collection，Decision，and Distribution(C2D2) Engine について説明する．

無線通信が Context に非常に依存することは理解されているが，一方で，通信性能を改善するために Context を利用するようなプロトコルがまだ完全には開発されていない．そこで，ネットワークスタックの多層にわたり既存の通信プロトコルと Context を接続する Context-aware のフレームワークの基礎を築いたという研究である．

車通信では，動的なチャンネル変更や端末の高い移動性が考えられるため，接続状態を維持でき，高スループットでの通信が可能であるような環境が望まれる．そこで，この研究では，アプリケーションが車通信における QoS を満たす上で利用可能な Context を判断し，生成されたデータをネットワークスタックの各層へ振り分けるフレームワークについて述べている．Context は図 2 に示すように Collection，Decision，Distribution を経由し，それぞれが適合する問題へ与えられる．すなわち，decision engine がセンサを用いて収集したデータの中から無線通信性能の改善に利用できるものを選択し，選択したデータを無線通信プロトコル上の適合する層へ分配するシステムである．本研究はこのうち，TCP/UDP 層における Context-aware Decision およびその Context の TCP/UDP 層への適用において利用可能な技術の確立を目指したものであるといえる．

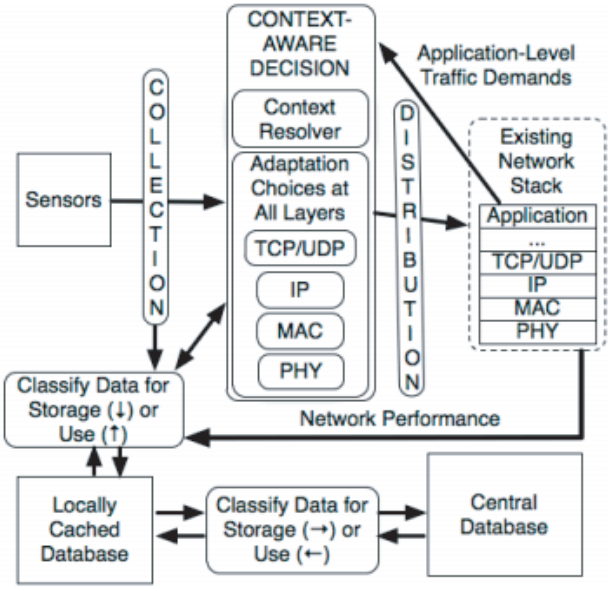

図 2　C2D2 システムのダイアグラム

3. TCP

3.1 TCP による輻輳制御

WLAN の国際標準として IEEE 802.11 が規定されており，IEEE 802.11 の上位層においては TCP により輻輳制御が実現されている．輻輳とはネットワークの混雑を表す言葉であり，ネットワーク上で多量のトラフィックが発生して，通常の通信

が困難になることをいう．TCP は，輻輳制御により輻輳を回避する仕様となっている．代表的な TCP として Tahoe，Reno，NewReno，Vegas 等があり，輻輳制御アルゴリズムは TCP によって様々な方式となっている．

TCP では，ウィンドウフロー制御をすることによって，輻輳の防止・早期回避を可能にしているが，一方で，輻輳回避フェーズにおける輻輳による cwnd サイズの減少幅が，後の線形増加時の増加幅と比較し大きいため，一度輻輳が起き cwnd サイズが下がると，cwnd サイズが増加するまでに時間がかかり，輻輳制御が原因となったスループットの低下が起こる場合がある．以下では，本稿で比較評価に用いた TCP である TCP Tahoe と TCP Reno の輻輳制御アルゴリズムについて説明する．

3.2 TCP Tahoe

TCP Tahoe は，スロースタートフェーズと呼ばれる，ACK セグメントを受信した分だけ cwnd サイズを増加させる動作を行う (式 (1)，if 条件式).

$$cwnd \longleftarrow \begin{cases} cwnd + 1 & \text{if cwnd} < \text{ssthresh} \\ cwnd + \frac{1}{cwnd} & \text{otherwise} \end{cases} \quad (1)$$

cwnd サイズ 1 でセグメントを送信すると，受信側から 1 つの ACK セグメントが送信されるため cwnd サイズは 2 となる．cwnd サイズが 2 のとき 2 つのセグメントを送信するため，受信側からは 2 つの ACK セグメントが送信され，cwnd サイズは 4 となる．このようにスロースタートフェーズでは cwnd サイズを 1,2,4,8,16,…と指数関数的に増加させていく．ここで，ssthresh はスロースタートから輻輳回避のフェーズへ移行する際の閾値とする．ssthresh 初期値は多くの実装で広告ウィンドウサイズが設定されるが，ある程度大きい値であれば任意である．また，輻輳が起こると ssthresh は以下のように更新される．

$$ssthresh \longleftarrow \frac{ssthresh}{2} \quad (2)$$

その後，cwnd サイズを 1 まで下げ，再びスロースタートフェーズを行う．やがて cwnd サイズが ssthreth に達すると輻輳回避フェーズへ移行し，cwnd サイズを式 (1)otherwise 条件式を用いて線形的に増加させる．ここで再度輻輳が起こると，再び式 (2) を用いて ssthreth を更新し，cwnd サイズを 1 まで下げ，スロースタートフェーズへ移行する．TCP Tahoe はこの動作を繰り返す．また，高速再送アルゴリズムが採用された点も TCP Tahoe の特徴である．

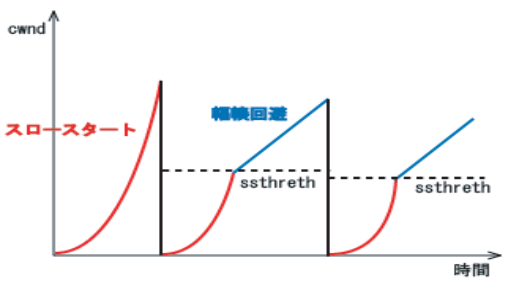

図 3　TCP Tahoe における cwnd の変化

3.3 TCP Reno

TCP Reno は TCP Tahoe がベースのアルゴリズムで，TCP Tahoe に対してエラーが起きても必ずしも cwnd サイズを 1 まで下げないように改良したものである．すなわち，始めはスロースタートフェーズで cwnd サイズの急速な増加を図り，その後，cwnd サイズが ssthresh に到達すると，輻輳回避フェーズへと移行する．輻輳回避フェーズでは，輻輳が起こると ssthresh を半分に更新し，cwnd サイズを ssthresh まで下げてから線形的に増加させ，輻輳が起こる度にこれを繰り返す (図 4)．また，パケットロスなどによりタイムアウトが起きると，cwnd サイズを 1 に変更しスロースタートフェーズの動作へと戻る．

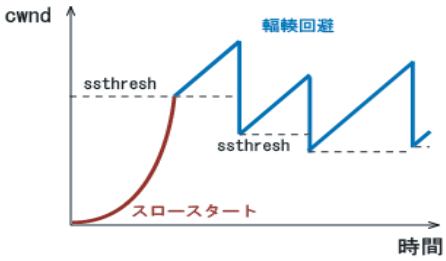

図 4　TCP Reno における cwnd の変化

3.4 TCP NewReno

TCP Reno では selective acknowledgement（SACK）がない場合に，複数パケットが損失すると高速再転送および高速リカバリアルゴリズムが有効に動作しない問題があった。この高速リカバリアルゴリズムを修整したものが TCP NewReno である．

4. OMNeT++

本研究ではネットワークシミュレータとして OMNeT++を使用している．OMNeT++は，Andras Varga によって開発されたオープンソースでオブジェクト指向の離散イベント型のネットワークシミュレーションフレームワークで，対応 OS は Unix および Windows である．OMNeT++におけるシミュレーションプログラムの構築手順を図 5 に示す．

OMNeT++のモジュールは C++言語で実装されており，高級言語を使用するモデル（NED：NEtwork Description）が組み立てられる．また，ソースコードを修正せずに，パラメータの変更だけでシミュレーション環境の構築ができ，Graphical User Interface（GUI）によるパラメータ設定も可能なため，ユーザはシミュレーションモデルを構築しやすい．また，シミュレーション状況をリアルタイムでアニメーション表示可能で，実装したモデルの動作把握やデバッグ処理が容易という特徴も持っている．OMNeT++は，有線および無線通信ネットワークのモデリングや，プロトコルのモデル化，ハードウェアの検証など様々な研究領域での使用に適している．

本研究において，シミュレーションモデルは OMNeT++上で動作する TCP/IP のシミュレーションパッケージである INET Framework を拡張した INETMANET を使用した．INET Framework は，UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11 などさ

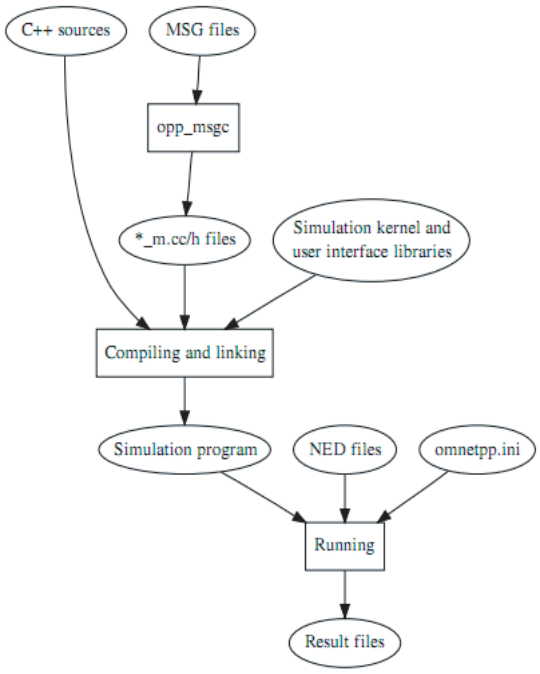

図 5　OMNeT++におけるシミュレーションプログラムの構築

まざまなプロトコルを OMNeT++上で動作させることができるシミュレーションモデルである．これに対し，INETMANET はアドホックネットワークなど INET Framework に含まれていない無線プロトコルに対応した仕様となっている．

5. 検 証 実 験

5.1 概　　要

本稿では，複数の端末 (クライアント) と 1 台のサーバが通信を行っており，クライアントから送信されたパケットが，サーバ側で Sink(パケット破棄) されるシナリオを想定した (図 6). このとき各クライアントは，WLAN でアクセスポイント (AP) に接続されており，AP から有線でルータを 2 台経由しサーバへパケットを送信している．

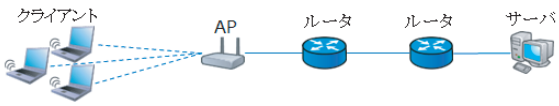

図 6　実験に用いたモデル

ここで, 輻輳制御が行われている TCP（TCP Tahoe，TCP Reno，TCP NewReno）と，輻輳制御が行われない TCP（No Congestion Control）の 4 種類の輻輳制御モデルを用い，それぞれのクライアント数に応じたスループットを測定した．TCP Tahoe，TCP Reno，TCP NewReno において，cwnd サイズは式 (1) および (2) が適用され動的に更新されながら通信が行われる．一方，No Congestion Control における cwnd サイズは，輻輳が起きても下げられず一定値を保つ設定で通信が行われる．このシナリオでは，無線規格として IEEE 802.11 を使用し，その他のパラメータは表 1 のように設定しシミュレーションを行った. 結果を図 7，図 8 に示す.

なお，図 7 の実験の際，RTT は 100msec に固定している．また，複数台での通信を想定しているためクライアント数は 2〜50 とした.

表 1　シミュレーションパラメータ

Send Bytes	2MByte
Advertised Window	8,356kByte
Data Rate	6Mbps
Carrier Frequency	2.4GHz
Maximum Segment Size	65,280Byte
Maximum Sending Power	20mW

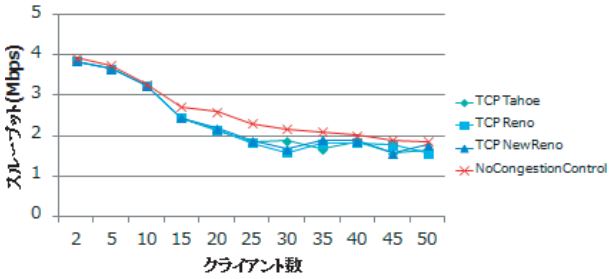

図 7　端末数に伴うトータルスループット

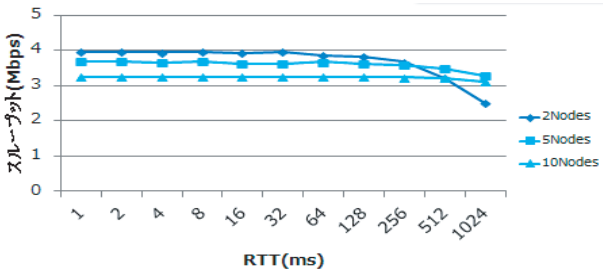

図 8　TCP NewReno の RTT に伴うスループット

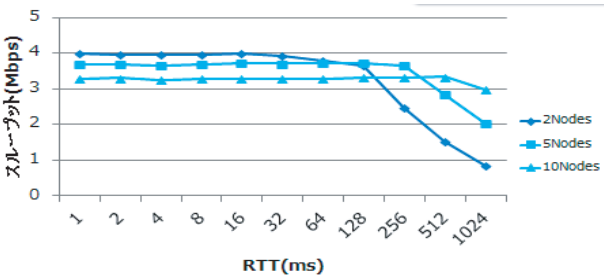

図 9　NoCongestionControl の RTT に伴うスループットの比較

5.2 考　　察

図 7 から，RTT が 100ms の場合では NoCongestionControl がクライアント数に関わらず最も通信性能が良いことがわかる．特にクライアント数が 15 い女王では NoCongestionControl の優位性がはっきり現れている．これらの性能差から，クライアント数が大きい場合に，TCP Tahoe や TCP Reno，TCP NewReno の輻輳制御が必ずしも効率良く機能しておらず，結果的に，cwnd サイズに制限をかけない NoCongestionControl の性能が高くなったものと考えられる．

次に，RTT に伴うスループットの変化を測定したところ，クライアント数が少ないほど，RTT の増加に伴いスループットが大きく低下する様子が見られた（図 8，図 9）．クライアント数の大きい場合，高遅延で必要となる大きな cwnd サイズを多数のノードで分担できるためスループットが低下しにくくなることから，これは妥当な結果であると言える．また，これらを比較すると，NoCongestionControl のスループットが高遅延環境においてより大きく低下している．これは，NoCongestionControl では cwnd サイズをある値に固定しており，遅延が小さい場合にはそれが十分なサイズであるが，遅延が大きい場合にこの設定が適切ではなく，通信を制限してしまったため，TCP NewReno の性能を下回ったと考えられる．

6. まとめと今後の課題

WLAN において遅延環境モデルを用い，一般に使用されている輻輳制御を行う TCP Tahoe，TCP Reno，TCP NewReno と，cwnd サイズを一定に保つ TCP である NoCongestionControl のそれぞれの端末数に伴うスループットをシミュレーションにより求めた．この結果から，従来の TCP ではクライアント数が大きい場合に端末数に応じた最適な通信が行われていない可能性があることがわかった．次に，TCP NewReno と NoCongestionControl の RTT に伴うスループットを比較した．高遅延環境では NoCongestionControl の性能が TCP NewReno を下回ったことから，NoCongestionControl で設定されている cwnd サイズの設定が，必ずしも最適ではないということがわかった．以上から，cwnd サイズを上手く設定できれば高遅延環境において従来の TCP より良い性能での通信が可能であると考えられる．

今回の結果から，NoCongestionControl のプラグラム内の cwnd サイズの設定方法を改変することで高遅延環境における通信性能を向上できると考えられるため，今後は，cwnd サイズを任意の値に固定できるような TCP を実装し，高遅延での通信性能の向上を実現したい．また，最終的には，cwnd サイズの設定に伴うスループットを測定することで，周辺の状況に応じた最適な通信設定の手法を検討したい．

謝　　辞

本研修を進めるにあたり，株式会社トヨタ IT 開発センターの Onur Altintas 氏に大変有用なアドバイスをいただきました．深く感謝いたします．

文　　献

[1] Daniel Salder, Anind K.Dey and Gregory D.Abowd, ”The Context Toolkit : Aiding the Development of Context-Enabled Applications”, In Proccedings of CHI’99,PIttsburgh,PA,May 15-20,1999,(to appear).ACM Press.

[2] Day,Anind K., ”Understanding and Using Context”(2001). Human-Computer Interaction institute.Paper 34. http://repository.cmu.edu/hcii/34

[3] Joseph Camp,Onur Altintas,Rama Vuyyuru,and Dinesh Rajan, ”Context-awere Collection,Decision,and Distribution(C2D2) Engine for Multi-Dimensional Adaptation in Vehicular Networks”, VANET ’11, pp.85-86, September 23, 2011

[4] 松本真紀子，Onur Altintas，西堀満洋，小口正人，”環境情報を利用したマルチプルアクセス手法の一検討”，情報処理学会第 73 回全国大会，1V-6，March 2011