WebSocket と WebRTC を用いた自動車テレマティクス サービスアーキテクチャの提案

2010SE151 成田　賢　2010SE180　Qiu　Binbin

指導教員：青山　幹雄

1　はじめに

本研究では，ユーザがサービスブロバイダや OS に依存せずにサービスを受けることのできる自動車テレマティクスサービスアーキテクチャ案を提案する．

2　研究背景

交通情報を地図上に表示できるサービスが普及している．しかし，リアルタイムでの正確な交通情報の配信は困難である．

ユーザの現在地に対して正確な交通情報をリアルタイムで提供できるサービスの構築が必要である．

3　研究課題

以下の点を研究課題とし，この課題を解決するアーキテクチャの提案をする．

(1)　OS やプラットフォームに依存しないサービス
SMD ではアプリケーションや特定の OS，車載器においては自社製品のみで動作するものでなくこれらに依存しないサービスを実現する．

(2)　P2P 通信を用いたリアルタイムな交通情報を実現するアーキテクチャ
渋滞情報などを提供するサービスは確立されているがリアルタイム情報がユーザに届かない場合がある．ユーザが欲しい交通情報をリアルタイムに提供するアーキテクチャの構造を提案する．

4　関連研究

4.1　HTML5

HTML5 は Web アプリケーションのプラットフォームやマルチメディアを表現する仕様が定義されている[2]．

4.2　WebSocket

ブラウザと Web サーバ間で双方向リアルタイム通信を実現する通信プロトコルである．HTTP 通信でブラウザと Web サーバのコネクションを確立する．確立したコネクション上では専用のプロトコルを用いて通信する[1]．WebSocket は TCP プロトコルを用いて通信をするため，時間制約よりも信頼性が重視される情報の通信に適している．

4.3　WebRTC

Web ブラウザ同士でリアルタイムなコミュニケーションを可能にする技術である．最初にサーバを用いて通信先を発見し，通信を確立する．その後は Web ブラウザ同士で P2P 通信を行う．そのため高速な通信が可能である[5]．WebRTC は UDP プロトコルを用いて通信をするため，時間制約が重視される動画や音声などの通信に適している．

5　アプローチ

HTML5 を用いることによって OS やプラットフォームに依存しないサービスのアーキテクチャの提案をする．WebSocket と WebRTC の 2 つの情報を同期させることでナビゲーションの情報と映像を重ね合わせることが可能になり，リアルタイムに情報を提供できる．

この技術を使うことによって車車間通信やスマートフォンとの連携が可能になる．ユーザから交通情報の提供によりサービスブローカが交通情報の補完ができる．

6　提案方法

6.1　提案アーキテクチャの構成

アーキテクチャの構成を図 1 に示す．

提案するアーキテクチャの構成はユーザのブラウザから情報送信，ナビ制御，映像制御を行う．情報送信はユーザのプローブデータの送信などを行う．ナビ制御はサーバから送られた交通情報や地図データをブラウザに表示させる．映像制御は他ユーザから取得した映像を処理する．

また，ナビゲーションの基本機能を保証するため処理には交通情報処理，地図データ処理，ユーザ情報処理を行う．交通情報処理はユーザから取得したプローブデータなどの処理を行う．地図データ処理はユーザの現在地を元に地図データを送る処理を行う．ユーザ情報処理はユーザの現在地の処理を行う．

サーバには WebSocket サーバ，ICE サーバを配置する．ICE サーバは他ユーザのグローバルアドレスの取得のために配置する．

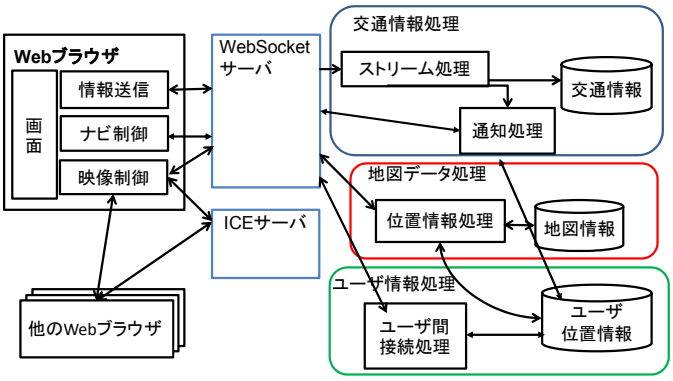

図 1　アーキテクチャの構成

6.2　提案アーキテクチャの詳細図

アーキテクチャについて「情報送信，ナビ制御」と「映像制御」の部分と分けて考える．

映像制御の仕組みの扱うデータの処理の詳細を図 2 に示す．

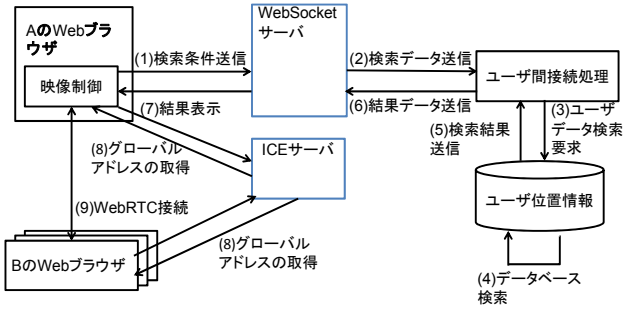

図 2 映像制御の詳細図

(1) ユーザは映像を取得したいユーザの条件を入力し，WebSocket サーバに検索条件のデータを送る.
(2) WebSocket サーバは検索条件データをユーザ間接続処理に送る.
(3) ユーザ間接続処理はデータ処理をした後ユーザ位置情報データベースに該当するユーザの検索をする.
(4) ユーザ位置情報データベースは条件にあったユーザをデータベース内で検索する.
(5) 検索終了後，検索結果をユーザ間接続処理に送る.
(6) ユーザ間接続処理は結果データを WebSocket サーバに送る.
(7) WebSocket サーバからユーザに検索結果を表示するデータを送る.
(8) ユーザは結果の中から接続をしたいユーザを選択し，P2P 通信をするのに必要である接続先のユーザのグローバルアドレスを取得するために ICE サーバに接続をする.
(9) ICE サーバからグローバルアドレスを取得した後 WebRTC による P2P 通信を行う.

次に情報送信とナビ制御の詳細を図 3 に示す.

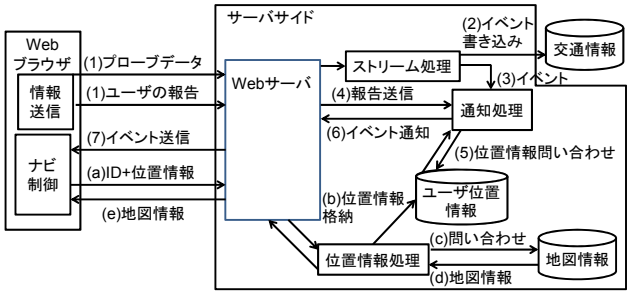

図 3 情報送信とナビ制御の詳細図

情報送信の処理について説明する.
(1) ユーザからのプローブデータとユーザからの事故情報などの報告を集計し，プローブデータや報告を Web サーバに送る.
(2) 収集したデータをストリーム処理し，イベントを交通情報データベースに書き込む.
(3) 何かしらの異常があれば，イベントを通知処理に渡す.
(4) ユーザからの報告を通知処理に送る.
(5) 情報を送るユーザを決定するためユーザ位置情報データベースに対象ユーザを検索する.
(6) 通知処理からイベント通知処理を Web サーバに送る.
(7) Web サーバからナビ制御に対してイベントを送信し，ナビに表示する.

地図情報取得の処理について説明する.
(a) ユーザから送られたユーザ ID と位置情報を位置情報処理に送る.
(b) 位置情報処理で処理をし，処理したデータから地図情報データベースに地図データを問い合わせする.
(c) 地図情報データベースから位置情報処理に地図情報を渡す.
(d) ユーザ位置情報データベースに対してユーザの位置情報を書き込む.
(e) 位置情報処理からナビ制御に対して地図情報を送る.

6.3 映像制御の振る舞い

映像制御の振る舞いを図 4 に示す.

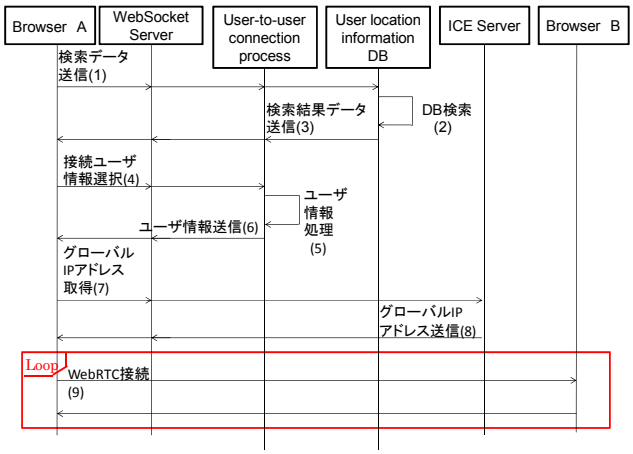

図 4 映像制御の振る舞い

振る舞いは以下になる.
(1) ブラウザ A から情報を取得する検索データをユーザ位置情報 DB に送信する.
(2) DB 内で該当する他ユーザを検索する.
(3) 検索結果データをブラウザ A に送信する.
(4) ブラウザ A は検索結果から接続するユーザを選択しユーザ間接続処理に送る.
(5) ユーザ間接続処理は選択されたユーザ情報を処理する.
(6) 他ユーザの情報をブラウザ A に送信する.
(7) ブラウザ A から ICE サーバにブラウザ B のグローバル IP アドレスの取得を依頼する.
(8) ブラウザ B のグローバル IP アドレスの取得ができ次第ブラウザ A に送る.
(9) WebRTC による P2P 通信を開始する.

7 提案アーキテクチャのプロトタイプ

提案するアーキテクチャに基づき，プロトタイプを作成した．提案するアーキテクチャの中核の部分は「映像制御」であるので，これをプロトタイプとして実装した.

7.1 利用シナリオ

「映像制御」の利用には「交通画像送受信サービス」，「リアルタイム交通状況確認サービス」の 2 つのシナリオが考えられる．またこのシナリオでのネットワーク環境は無線通信を考えている．

「交通画像送受信サービス」は他ユーザから送信された画像をサーバから取得し，ユーザの現在地位置情報を元にデータセンタが交通情報(渋滞，交通事故，工事，道路規制)をユーザに送信する．WebSocket によって実現されているのでユーザ側からの要求がなくてもデータセンタ側からの交通情報の自動更新が可能である．

「リアルタイム交通状況確認サービス」は半径 1km に存在する他ユーザの情報を取得し，ユーザが接続したいユーザを選択する．選択した後他ユーザの Web カメラと WebRTC による P2P 通信を行いユーザのブラウザに Web カメラの映像を表示する．これによってユーザから離れた道路情報をリアルタイムに取得することができる．

7.2 実装環境

プロトタイプがプラットフォームに依存しないことを検証するために，異なる OS 上に実装し，動作確認を行う必要がある．本研究では Windows 7, WindowsXP, Mac OS X の3つの OS 上でプロトタイプが正しく動作できるかを確認した．実装に利用するブラウザとして Chrome を選択した．また，ユーザ B の車から取得する交通状況の映像として，Web カメラで取得される時計の映像を利用した．実装環境を表 1 に示す[3]．

表 1 プロトタイプの実装環境

OS	Windows7	Windows XP	MacOS X
メモリ	2GB	1GB	8GB
プロセッサ	2.53GHz Intel(R) Core(TM) 2Duo	1.31GHz Genuine Intel(R)	2GHz Intel Core i7
Web カメラ	USB2.0 Web カメラ		FaceTime HD カメラ
ブラウザ	Chrome 31.0.1650.63		
WebSocket サーバ	Node.js v0.10.24		
STUN サーバ	stun.l.google.com:19302		
ライブラリ	easyRTC.js		
インターネット	Au ひかり(下り最大 70Mbps、上り最大 30Mbps)		

サーバ側の処理は以下のようになる．

(1) サーバの環境設定を行い，easyRTC 接続機能をロードする．
(2) ファイルシステムコアモジュール fs, http サーバコアモジュール http, ウェブフレームワーク外部モジュール espress, WebSocket 外部モジュール Socket.io, ログ解析モジュール winston をロードする．
(3) Http サーバと Socket サーバを作成する(Socket サーバは Http サーバからアップグレードされる)．
(4) 接続用の Socket を生成し，サーバとクライアント間のデータを送受信する．

7.3 自動車テレマティクスサービスの実装

Node.js を利用し，サーバプログラムを作成する．Node.js はサーバサイドの JavaScript インタープリタで「シングルスレッドベースの非同期処理環境」という特徴を持つ[4]． 作成した自動車テレマティクスサービスの構成を図 5 に示す．

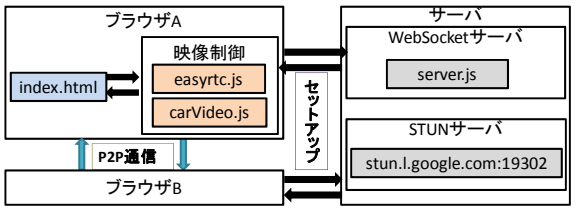

図 5 自動車テレマティクスサービスの構成図

7.4 プロトタイプの振る舞い

プロトタイプのシーケンス図を図 6 に示す．

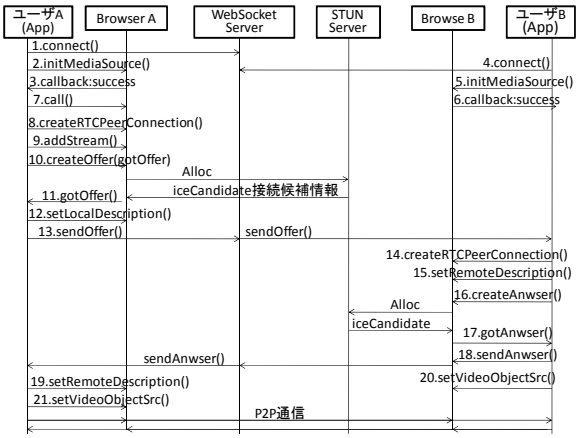

図 6 プロトタイプのシーケンス図

またユーザ A がサービス利用開始からユーザ B との P2P 通信終了までの時間を図 7 に示す．

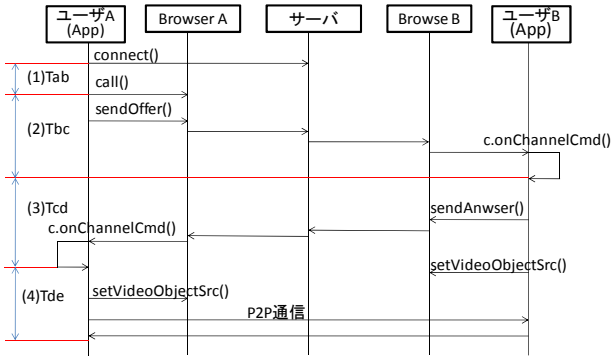

図 7 サービス利用時間の流れ

処理時間を以下のように測定する．

(1) Tab: ブラウザ A がサーバに接続(connect()実行)後，ユーザ B の ID ボタンを押す(call()実行)前の時間．
(2) Tbc: ユーザ B の ID ボタンを押され(call()実行)，ブラウザ A からブラウザ B にオファー到着まで(c.onChannelCmd()実行)の時間．
(3) Tcd: ブラウザ B がオファーを受信し，アンサーを作成する．ブラウザ A にアンサー到着した後，ブラウザ間 P2P 通信開始まで(ブラウザ A が setVideoObjectSrc()実行)の時間．
(4) Tde: ブラウザ間 P2P 通信の時間．

図 7 で示すように，P2P セットアップ時間 T は T=Tbc+Tcd である．

8 プロトタイプに基づくアーキテクチャの評価

本研究では現在のカーナビゲーションサービスの「プラットフォームに依存」と「リアルタイムな情報の提供が不

可能」の2つの課題に着目し，作成したプロトタイプを利用し，提案する自動車テレマティクスサービスのアーキテクチャの評価をした．

8.1　プラットフォーム非依存性の評価

本研究では，WebRTC のライブラリ easyRTC と Node.js を用いたプロトタイプを 3 つの OS 上で実装し，動作確認を行った．ブラウザは Google の Chrome を選択した．詳細は表 2 に示す．

表 2 プロトタイプが利用するブラウザと OS の詳細

OS	Windows 7	Windows XP	MacOS X
プロセッサ	2.53GHz Intel(R) Core(TM) 2Duo	1.31GHz Genuine Intel(R)	2GHz Intel Core i7
ブラウザ	Chromeバージョン 31.0.1650.63		

プロトタイプを実装した結果，3 つの OS でプロトタイプが問題なく動作することにより，提案する自動車テレマティクスサービスのアーキテクチャがプラットフォームに依存しないことを確認した．

8.2　情報提供の測定

提案する自動車テレマティクスサービスのアーキテクチャでは，インターネットの通信状況により，通信の遅延が生じる．P2P 通信前の STUN サーバと通信による遅延と WebSocket サーバと通信による遅延，P2P 通信後の遅延が想定される．作成したプロトタイプは Node.js でローカル WebSocket サーバを配置した．WebSocket サーバはサービスの提供とブラウザ間の通信に利用される．また，Google が提供する STUN サーバを用いた NAT 越えを行うため，Au ひかり(下り最大 70Mbps，上り最大 30Mbps)を利用している．

各 OS でプロトタイプを実行した場合の P2P 通信開始のセットアップ時間の測定結果を表 3 に，異なる OS 上の P2P 通信前の準備時間 T の比較を図 8 に示す．

表 3　P2P のセットアップ時間の測定結果

	Windows XP			Windows 7			MacOS		
	Tbc	Tcd	T1	Tbc	Tcd	T2	Tbc	Tcd	T3
1	0.062	0.496	0.558	0.035	0.198	0.233	0.019	0.130	0.149
2	0.085	0.543	0.628	0.061	0.266	0.327	0.015	0.122	0.137
3	0.086	0.557	0.643	0.058	0.270	0.328	0.017	0.104	0.121
4	0.069	0.695	0.764	0.037	0.291	0.328	0.017	0.096	0.113
5	0.076	0.694	0.770	0.046	0.328	0.374	0.017	0.109	0.126
6	0.055	0.527	0.582	0.035	0.324	0.359	0.016	0.181	0.197
7	0.065	0.609	0.674	0.083	0.229	0.312	0.017	0.116	0.133
8	0.057	0.581	0.638	0.051	0.245	0.296	0.016	0.111	0.127
9	0.064	0.436	0.500	0.042	0.239	0.281	0.015	0.159	0.174
10	0.068	0.544	0.612	0.043	0.284	0.327	0.018	0.111	0.129
平均	0.068	0.568	0.636	0.049	0.267	0.316	0.016	0.123	0.140
標準偏差	0.010	0.077	0.079	0.014	0.041	0.039	0.001	0.025	0.024

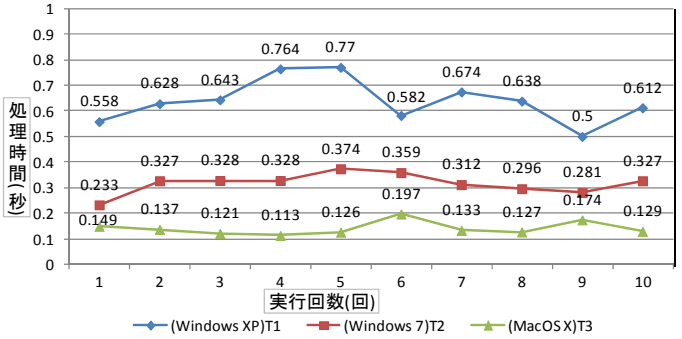

図 8　P2P のセットアップ時間の比較

8.3　情報提供の評価

計測結果を分析すると，ブラウザ A 上にユーザ B の ID ボタンを押すと，ブラウザ B にオファーが到着までの時間 Tbc の変動が小さいことが分かった．これはセットアップを除き全ての通信がローカル環境にある WebSocket サーバを利用しているため，インターネットによる影響がないと考えられる．

ブラウザ B がアンサーを作成し，ブラウザ A にアンサー到着した後，ブラウザ間 P2P 通信開始までの時間 Tcd の変動が大きいことも分かった．これは P2P 通信をする場合 NAT 越えを行う必要があり，ブラウザが外部にある STUN サーバに問い合わせる．この問い合わせは Windows XP，Windows 7 では 18 回．MacOS X では 10 回となった．この問い合わせの違いの原因としては Windows ではファイアフォールにより UDP プロトコルだけでなく TCP プロトコルを用いて問い合わせをしているため問い合わせ回数が増加したと考えられる．

Tbc と Tcd の比較をした場合，SendAnwer()の生成時間によって Tcd が変動していると考えられる．Tbc の平均は Windows XP では 0.068s，Windows 7 では 0.049s，MacOS X では 0.016s となった．短時間でのセットアップで P2P 通信による情報提供が可能である．現在提供されている自動車テレマティクスサービスと比較した場合多くのサービスは交通状況の送信間隔を分単位で設定している．提案アーキテクチャの場合 100ms 以内でセットアップができ，その後 P2P 通信による情報交換が可能な点からテレマティクスサービスに適応可能と考えられる．

9　今後の課題

今後の課題として，以下の点が挙げられる．

(1)　ナビ制御，情報送信の実装
(2)　セキュリティの考慮
(3)　スマートフォン上での実装

10　まとめ

現在の自動車テレマティクスサービスは，他社のサービスを受けることやリアルタイムな情報をユーザが把握することが困難である．本研究では HTML5 を用いた Web ブラウザによる Web サービスと HTML5 の技術の 1 つである WebRTC と WebSocket を同期することによりリアルタイムな交通情報をユーザに提供するアーキテクチャ案を提案した．

参考文献

[1]　IETF, The WebSocket Protocol, 2011, http://tools.ietf.org/html/rfc6455.
[2]　小松 健作，徹底解説 HTML5 マークアップガイドブック 最終草案対応版，秀和システム，2011.
[3]　Priologic Software Inc, easyRTC, 2013, http://easyrtc.com/.
[4]　清水 俊博，他，サーバサイド JavaScript: Node.js 入門, ASCII, 2012.
[5]　W3C,WebRTC 1.0: Real-Time Communication between Browsers,2013. http://www.w3.org/TR/webrtc/