Vol.2014-DPS-158 No.2
Vol.2014-CSEC-64 No.2
2014/3/6

XMLHttpRequest フッキングによる既存 Web アプリケーションの WebSocket 移行方式の検証と評価

加島隆博　　羽藤淳平　　虻川雅浩

XMLHttpRequest による Ajax 技術が Web に導入されたことで，Web ブラウザとサーバ間でページ遷移を伴わない HTTP 通信が可能になり，リアルタイム性の高い Web アプリケーションの実現が可能になった．更に，プッシュ配信や通信効率等の HTTP が抱える問題を解決した新たな通信プロトコルとして WebSocket が策定され，XMLHttpRequest に代わって使われ始めている．

WebSocket の課題として，XMLHttpRequest と API が異なるため，コンテンツへのコードの変更量が多いことが挙げられる．また，既存の HTTP サーバも WebSocket に対応しなければならない．

この課題に対し著者らは，2 段階の移行方式を検討した．段階 1 の特徴は，コンテンツに XMLHttpRequest を置き換えるライブラリを導入することと，サーバに WebSocket と HTTP の変換サーバを導入することで，Web アプリケーションに対する変更を最小限に抑えられることである．段階 2 では更に，既存の HTTP サーバだけを WebSocket に対応するだけで，変換サーバを廃し，サーバをより効率良く動作させることができる．

評価の結果，1 秒間に Web ブラウザがサーバから応答を受け取った回数は，移行前に比べ，段階 1 で 1.7 倍，段階 2 で 2.9 倍となった．これにより，XMLHttpRequest を使用する既存の Web アプリケーションへの変更を最小限に抑えつつ，よりリアルタイム性の高い Web アプリケーションの実現が可能であることが分かった．

WebSocket Migration Method of Existing Web Applications by Hooking XMLHttpRequest

TAKAHIRO KASHIMA　JUMPEI HATO　MASAHIRO ABUKAWA

Since the Ajax technologies by XMLHttpRequest enabled web pages to communicate with servers without changing the pages, web pages can display real-time information. Moreover, the WebSocket protocol was established to solve the problems of HTTP, such as push-delivery and efficiency, and is utilized recently instead of XMLHttpRequest.

The WebSocket API and protocol is different from XMLHttpRequest. In order to migrate an existing web application from XMLHttpRequest to WebSocket, we have to change a lot of the existing programs on both the contents and the server, so that the programs can support the WebSocket.

In this paper, we propose a two-step migration method. The first step introduces a library that hooks the native XMLHttpRequest on the contents, and introduces a transform server that transforms the HTTP and WebSocket bidirectionally. The first step does not need to change the existing programs. Furthermore, the second step adapts only the existing HTTP server for WebSocket. In the second step, the web application can work more efficiently since the transform server used in the first step can be removed.

As an evaluation result, the number of responses sent from the server and received by the web browser per second, increased 1.7 times in the first step, and 2.9 times in the second step, compared with the original web application. This result proved that our method can migrate an existing web application from XMLHttpRequest to more efficient WebSocket, without a lot of changes of the programs.

1. はじめに

XMLHttpRequest による Ajax (Asynchronous JavaScript + XML) 技術が Web に導入されたことで，Web ブラウザとサーバ間でページの再読み込みや遷移を伴わない通信が可能になり，リアルタイム性の高い Web アプリケーションの実現が可能になった．また，XMLHttpRequest が使用するプロトコルである HTTP (HyperText Transfer Protocol) にはプッシュ配信や通信効率等に課題があったが，これらの HTTP の課題を解決した新たなプロトコル及び API として WebSocket が策定され，XMLHttpRequest に代わって使われ始めている．

従来の XMLHttpRequest に比べて利点の多い WebSocket であるが，両者のプロトコルや API は異なるものである．そのため，XMLHttpRequest を使用する既存の Web アプリケーションを WebSocket に移行するには，Web ブラウザ側及びサーバ側のプログラムを変更する必要がある．大規模な Web アプリケーションにおいて，この変更は容易ではない．

そこで著者らは，既存の Web アプリケーションへの変更を最小限に抑えつつ，WebSocket の利点を活かせる移行方式を検討した．この方式は，Web ブラウザの XMLHttpRequest を置き換えること，及び，サーバマシンに変換サーバを導入することを特徴とする．

本稿では XMLHttpRequest と WebSocket の概要について述べ，次に本方式の詳細と評価結果を述べ，最後に考察を述べる．

三菱電機株式会社 情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation

Vol.2014-DPS-158 No.2
Vol.2014-CSEC-64 No.2
2014/3/6

2. XMLHttpRequest と WebSocket

本章では XMLHttpRequest と WebSocket の概要を述べる．

XMLHttpRequest は，2005 年頃までに多くの Web ブラウザが対応した API であり，Ajax と呼ばれる非同期通信を可能にした．従来，Web ページ内の情報を更新するにはページを再読み込みする必要があったが，XMLHttpRequest により再読み込みの必要を無くし，よりリアルタイムに情報を更新できるようになった．

XMLHttpRequest は通常，通信プロトコルとして HTTP を使用する．本稿では，XMLHttpRequest が通信するサーバを HTTP サーバと称する．XMLHttpRequest を活用する Web アプリケーションの観点から HTTP を考察すると，主に以下の問題点が挙げられる．

(1) Web ブラウザ側を起点とするリクエスト・レスポンス型であるため，HTTP サーバ側から任意の時点で Web ブラウザに情報を配信する（プッシュ配信）には，ロングポーリング等の手法を使用する必要がある．
(2) ヘッダは文字列により表現されるため，人間にとって理解しやすいが，コンピュータによる解析処理に時間がかかる．
(3) リクエストとレスポンスに多くのヘッダ情報が付加されるため，通信負荷や遅延が大きい．
(4) 同一のサーバに対する同時接続数に上限がある．
(5) Keep-Alive に対応していない HTTP サーバでは，リクエスト・レスポンスごとに接続が切断されるため，繰り返し処理を行うには再接続する必要がある．接続には TCP のハンドシェイク処理等が含まれるため，通信負荷が大きい．

一方，WebSocket は，これらの問題点を解決するために策定された JavaScript の API[1]，及び，通信プロトコル[2]である．本稿では，WebSocket のサーバを WebSocket サーバと称する．WebSocket は以下の特徴を持つ．

(1) リクエスト・レスポンス型でない．一度 Web ブラウザとサーバの接続が確立すれば，どちらからでも任意の時点でデータを送信できる．
(2) ヘッダはバイナリデータにより表現されるため，コンピュータで解析しやすい．
(3) リクエスト及びレスポンスに不必要なヘッダが付加されることはない．WebSocket の送受信に必要なヘッダも小さくなるように最適化されている．
(4) 同一のサーバに対する同時接続数に上限がない．
(5) データを送受信するたびに接続を切断する必要がない．持続的に接続可能である．

これらの特徴により，WebSocket は XMLHttpRequest よりも優れていると言える．XMLHttpRequest を使用する既存の Web アプリケーションを WebSocket に移行することで，性能の向上が期待できる．

3. 移行に伴う課題

前章では XMLHttpRequest に比べて WebSocket が優れていることを述べたが，本章では XMLHttpRequest を使用する既存の Web アプリケーションを WebSocket に移行する際の課題について述べる．

既存の Web アプリケーションを XMLHttpRequest から WebSocket に移行することで，より低負荷でリアルタイム性の高い Web アプリケーションの実現が可能となる．しかし，XMLHttpRequest と WebSocket は，Web ブラウザが提供する JavaScript の API や，Web ブラウザとサーバ間の通信プロトコルは互換性がなく，異なるものである．従って，XMLHttpRequest から WebSocket に容易に移行できるものではない．移行するには，Web ブラウザ側で処理するコンテンツ (HTML による文書構造や JavaScript によるプログラム)，及び，サーバのプログラムの両方を変更する必要がある．

まずはコンテンツ側の問題から述べ，次にサーバ側の問題について述べる．

3.1 コンテンツ側

Web ブラウザはサーバから取得したコンテンツにより処理を行う．このコンテンツに記述されているプログラムは JavaScript であり，XMLHttpRequest と WebSocket の API が提供されているが，両者の API は異なる．XMLHttpRequest の主な API を表 1 に，WebSocket の主な API を表 2 に示す．

表 1　XMLHttpRequest の主な API
Table 1　Basic APIs of XMLHttpRequest

API	説明
XMLHttpRequest()	コンストラクタ
open(メソッド, URL)	URL を開く
send([データ])	リクエストを送信する
responseText	レスポンスを取得する
onreadystatechange	通信状態が変化した際に呼ばれるイベント

表 2　WebSocket の主な API
Table 2　Basic APIs of WebSocket

API	説明
WebSocket(接続先)	コンストラクタ
send(データ)	サーバにデータを送信する
onopen	サーバと接続した際に呼ばれるイベント
onmessage	サーバからメッセージを受信した際に呼ばれるイベント

表 1 と表 2 に示すように，両者の API は異なっている．例えば，XMLHttpRequest で HTTP サーバからのレスポンス

を受信するには，onreadystatechange イベントを受け，responseText プロパティを参照すれば良い．一方 WebSocket では，WebSocket サーバからのメッセージが来たことを onmessage イベントで受け，同イベントの引数を参照する．また，WebSocket では，onopen イベントにより WebSocket サーバとの接続を確立したことを確認した後でなければ，send メソッドでデータを送信することはできない．

このように API が異なるため，コンテンツを WebSocket に移行するには, XMLHttpRequest が使用されている箇所のコードの全てを変更する必要がある．大規模な Web アプリケーションの場合, XMLHttpRequest が使用されている箇所は多数に上るため，この変更作業は困難である．

3.2 サーバ側

前述のように, XMLHttpRequest は HTTP により通信を行うが，HTTP と WebSocket にプロトコルの互換性は無く，既存の HTTP サーバから WebSocket に対応したサーバを導入する必要がある．

また，Web ブラウザが非同期通信で HTTP サーバから取得するレスポンスデータは，ニュース記事，株価，雨量計等のセンサ値等，動的に変化しうるデータであることが一般的である．このような動的なレスポンスデータの生成を実現するため，既存の HTTP サーバ側の処理は，CGI 等のプログラムによって，リクエスト内容を解析し，レスポンスを生成するという処理になっていることが多い．このような HTTP サーバ側のプログラムは，プロトコルとして HTTP を念頭に置いた設計になっている可能性がある．例えば HTTP のヘッダ情報にデータを載せることは一般的である．そのため，プロトコルの異なる WebSocket に移行するには，プログラムの多くを変更する必要が生じる恐れがある．

以上のように，XMLHttpRequest と WebSocket は異なる API，及び，プロトコルであるため，移行は困難である．そこで著者らは，既存の XMLHttpRequest を使用する Web アプリケーションに対して，コンテンツ，及び，サーバ側のプログラムへの変更を最小限に抑えつつ，WebSocket を活用できる移行方式を検討した．

4. 関連研究

本章では XMLHttpRequest と WebSocket に関する関連研究を概観する．

文献[3]では, 従来 XMLHttpRequest により通信する REST (REpresentational State Transfer) を WebSocket で通信するというプロトコル，及び，プログラムが提示されている．WebSocket にすることで性能改善が期待できるが，サーバ側は JAX RS/Jersey というフレームワークを使用しなければならないことや，専用のライブラリを使用してコンテンツを作成する必要がある. 既存の Web アプリケーションがこれらのフレームワークやライブラリを使用していれば，移行は容易であると言えるが，そうでない場合は容易ではない．

文献[4]では，モバイル端末において，通信速度の遅いモバイル端末とサーバ間の通信を，HTTP から WebSocket にすることにより, 通信速度を向上したことが示されている．この方式では Web アプリケーションに対する変更は無い一方で，個々のモバイル端末にプロキシ型のクライアントソフトウェアを導入しなければならない．端末が多数である場合，個々の端末にソフトウェアを導入することは難しい．

5. 提案方式

本章では本提案方式について述べる．

本提案方式の特徴は，コンテンツ側，及び，サーバ側それぞれに対し，XMLHttpRequest/HTTP と WebSocket の変換層を導入することを特徴とする．また，サーバ側の負荷を考慮し，サーバ側のみ 2 段階の移行方式とした．クライアント側は段階 1，2 共に違いはない．移行段階を表 3 に示す．

表 3　提案方式の移行段階
Table 3　Migration Steps of Proposed Method

段階	コンテンツ側	サーバ側
段階 1	XMLHttpRequest を置き換える．	変換サーバを導入する．
段階 2		既存 HTTP サーバを WebSocket サーバに変更し，変換サーバを廃止する．

以下で段階 1 と 2 で共通のコンテンツ側について詳細を述べ，その後，段階 1 と 2 について詳細を述べる．

5.1 コンテンツ側

コンテンツ側では，Web ブラウザの XMLHttpRequest クラスを，独自のクラスに置き換える作業を行う．コンテンツは JavaScript によるプログラムにより処理を行うが，この JavaScript は，Web ブラウザがビルトインで提供しているクラスであっても，容易に置き換え可能である．JavaScript の XMLHttpRequest クラスを置き換えることで，コンテンツに対し Web ブラウザが提供している本来の XMLHttpRequest を使用させず，代わりに本方式独自の XMLHttpRequest クラスの使用を強制できる．この独自の XMLHttpRequest クラスでサーバとの通信を WebSocket で行い，かつ，見かけ上は XMLHttpRequest と同様に振る舞うことで，コンテンツを大幅に変更せず，WebSocket を活用できる．この XMLHttpRequest の置き換えを行うプログラムを 1 つのソースコードにまとめることで，既存のコンテンツには以下のような 1 行を追加するだけで済む．

```
<script type="text/javascript" src="xhr-hook.js"></script>
```

5.2 段階 1

段階 1 では, 前述の XMLHttpRequest の置き換えに加え, 既存の HTTP サーバが稼働しているサーバマシンに変換ソフトウェアを導入する. 図 1 に段階 1 のシステム構成図を示す.

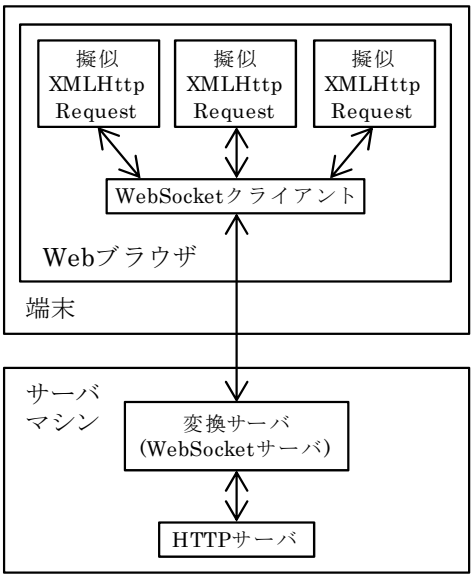

図 1　段階 1 のシステム構成
Figure 1　System Structure of First Step

コンテンツ側で置き換えられた XMLHttpRequest は, WebSocket クライアントとやり取りを行う. WebSocket クライアントは Web サーバごとに作成され, サーバマシンの変換サーバと WebSocket によって通信を行う. WebSocket クライアントは XMLHttpRequest によりリクエスト内容を受け取ると, リクエスト内容を直列化し, 変換サーバにリクエストを送信する. 変換サーバに送信するデータは以下の通りである.

- メソッド名 (GET や POST 等)
- リクエスト URL
- 任意に設定されたヘッダ
- POST メソッドの場合, リクエストボディ

変換サーバはリクエストを受け取ると, HTTP サーバにリクエストを発行し, レスポンスを受け取る. そして, WebSocket クライアントを通じて XMLHttpRequest にレスポンスを転送することで, コンテンツ側がレスポンスを受信する. レスポンスデータは以下の通りである.

- ステータスコード (200 等)
- HTTP サーバから返されたヘッダ
- レスポンスボディ

すなわち, 端末とサーバマシンの通信は WebSocket による高効率の通信を行うことで, WebSocket の利点を活用できる. 端末側の WebSocket クライアント及びサーバマシン側の変換サーバを介すことにより, Web アプリケーションに対する変更を最小限に抑えている.

また, Web ブラウザからサーバマシンに送信するリクエストは, 移行前よりもデータ量を削減できる. 通常の XMLHttpRequest を使用すると, User-Agent や Accept-Language 等, Web ブラウザが暗黙的にヘッダを付加する. これらは多くの Web アプリケーションにとって必要のないヘッダでありながら, セキュリティ上多くの Web ブラウザでは送信しないようにする方法が無い. 本方式では, コンテンツによって任意に設定されたヘッダのみ送信するため, リクエストヘッダを削減できる.

変換サーバと HTTP サーバ間は, 従来通り HTTP による通信を行うことで, HTTP サーバの変更点はない. 変換サーバと Web サーバはどちらもサーバマシン内のプロセスであるため, 両者の通信はプロセス間通信であり, 物理的なネットワークの通信と比べて高速である. 従って, 通信面での TCP や HTTP の非効率性は無視できると考える.

なお, WebSocket クライアントと変換サーバ間の通信は, HTTP のリクエストとレスポンスを WebSocket によって送受信するが, この通信内容の表現には, いくつかの形式が考えられる. 例えば JSON (JavaScript Object Notation) により, リクエストやレスポンスの内容をテキスト形式のデータ記述言語で送受信する形式が考えられる. この形式は JavaScript と親和性が高く, また, 人間が読む際も理解しやすいが, データ量が大きくなると JSON の作成や, 解析に時間がかかると考えられる. そこで今回は, 単純でコンピュータにとって扱いやすいバイナリ形式のデータを送受信することとする.

5.3 段階 2

段階 1 の方式では, サーバマシンに WebSocket と HTTP の変換を行う変換サーバを導入する必要があった. この変換処理のため, サーバマシンの負荷は移行前よりも高くなることが予想される.

そこで, 段階 2 ではサーバマシンの負荷を下げることと, 通信をより高速にするため, HTTP サーバのみを WebSocket サーバに変更する. これにより, 変換サーバを介さずとも WebSocket クライアントと WebSocket サーバが直接 WebSocket で通信できる.

段階 2 のシステム構成図を図 2 に示す. 段階 1 から段階 2 に移行するには, HTTP サーバのプログラムを変更する必要があるが, コンテンツには段階 1 の方式から一切の変更を必要としない.

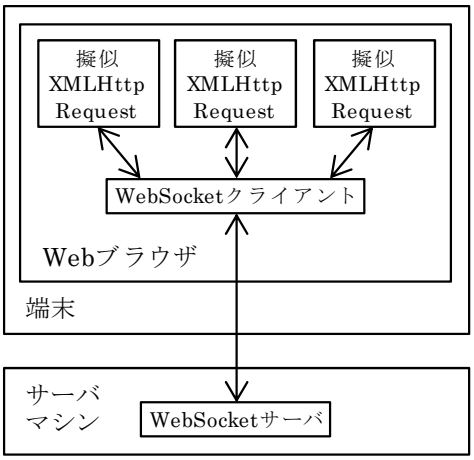

図 2　段階 2 のシステム構成
Figure 2　System Structure of Second Step

6. 評価

本提案方式の有効性を示すため，Web ブラウザとサーバ間で通信を行い，WebSocket への移行前と本提案方式を比較した．まず評価方法について述べ，次に結果を述べる．

6.1 評価方法

Web ブラウザからサーバにリクエストを送信し，レスポンスを受信するまでの流れを，1 回の応答とする．移行前，本提案方式の段階 1，及び段階 2 に対し，この処理を 1 分間繰り返し行い，以下の項目を算出した．

- 1 秒あたりの平均応答回数
- Web ブラウザとサーバの平均処理時間
- リクエスト及びレスポンスの通信量
- 1 応答あたりの平均通信時間

評価方法の処理の流れを図 3 に示す．

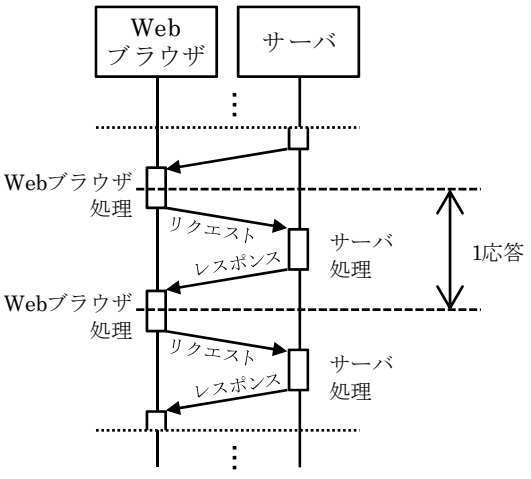

図 3　評価方法の処理の流れ
Figure 3　Sequence of Evaluation

サーバがリクエストを受信しレスポンスを送信するまでの間や，Web ブラウザがレスポンスを受信し次のリクエストを送信するまでの間に待機時間は設けない．

Web ブラウザがサーバに送信するリクエストは，サーバの URL に対する単純な GET メソッドである．一方，サーバが Web ブラウザに送信するレスポンスは，メッセージボディに 4 KB のデータを持つレスポンスである．

端末とサーバマシンはそれぞれ 1 台であり，スイッチングハブを介して 1000BASE-T のイーサネットで接続した．ネットワークの構成図を図 4 に示す．また，サーバマシンと端末は同じ性能のコンピュータである．使用したハードウェアやソフトウェアを表 4 に示す．

図 4　ネットワーク構成
Figure 4　Network Configuration

表 4　各装置のハードウェアとソフトウェアの仕様
Table 4　Hardware and Software Specifications of Computers

CPU	Intel Core i7 2.5GHz
メモリ	4 GB
OS	Windows 7 64bit
Web ブラウザ	Google Chrome 32
サーバ S/W	Node.js v0.10.24
WebSocket モジュール	ws 0.4.31

6.2 評価結果

1 秒あたりの平均応答回数を図 5 に示す．

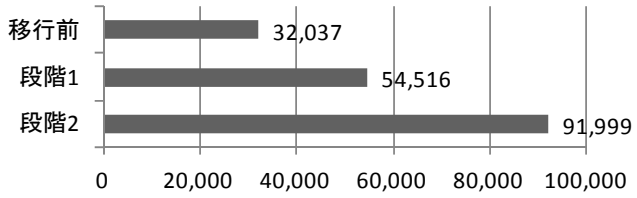

図 5　1 秒あたりの平均応答回数
Figure 5　Average Response Counts per Second

Web ブラウザ及びサーバの平均処理時間を，表 5 と図 6 に示す．

表 5　平均処理時間
Table 5　Average Processing Times

	Web ブラウザ	サーバ	計
移行前	1,141	404	1,545
段階 1	280	533	813
段階 2	227	159	386

単位: μ 秒

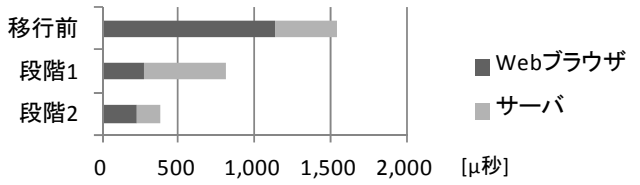

図 6　平均処理時間

Figure 6　Average Processing Times

リクエストとレスポンスの通信量と平均通信時間を表 6 に示す．なお，通信量は HTTP もしくは WebSocket レイヤのデータ量であり，平均通信時間はリクエストとレスポンスの合計時間である．

表 6　通信量と平均通信時間

Table 6　Traffic Volumes and Average Communication Times

	通信量[Byte]				通信時間[μ秒]
	リクエスト	レスポンス		計	
		ヘッダ	ボディ		
移行前	437	217	4,096	4,750	328
段階 1	124	244		368	287
段階 2		18		142	265

7. 考察

本章では前章の評価結果を考察する．

図 5 の 1 秒あたりの平均応答回数によると，移行前に比べ段階 1 は 1.7 倍，段階 2 は 2.9 倍の応答回数となり，本提案方式により WebSocket に移行することで，性能を向上できることが分かる．

性能が大きく改善した処理は，Web ブラウザの処理である．表 5 及び図 6 が示す平均処理時間によると，Web ブラウザの処理時間は移行前に比べ，段階 1 で 25%，段階 2 で 20%の時間に短縮できたことが分かる．これは，HTTP に比べ WebSocket は処理性能を考慮してプロトコルが設計されており，リクエストの生成，及び，レスポンスの解析処理に負担が少ないからであると考える．なお，段階 1 のサーバの処理時間は，変換サーバが導入されたことにより移行前よりも増えているが，サーバ側で増加した時間よりも Web ブラウザ側の処理時間の減少の方が多いため，全体の処理時間は移行前よりも改善している．

表 6 の通信量と平均通信時間によると，本提案方式では XMLHttpRequest が付加する不要なヘッダを除外できるため，リクエストの通信量を削減できている．更に，段階 2 ではレスポンスに不要なヘッダも除外できるため，通信量を削減できている．しかし，移行前と比較した段階 1 や段階 2 の処理と通信の削減時間と貢献率をまとめた表 7 を見て分かる通り，本方式による処理時間の改善に比べて，通信量の削減は低いことが分かる．

表 7　処理時間と通信時間の削減

Table 7　Reductions of Processing and Communication Times

	削減時間[μ秒]			貢献率	
	処理	通信	計	処理	通信
段階 1	732	41	773	94.7%	5.3%
段階 2	1,159	63	1,222	94.8%	5.2%

8. おわりに

本稿では，XMLHttpRequest を使用する既存の Web アプリケーションを WebSocket に移行する方式について述べた．本方式の特徴は，XMLHttpRequest を独自のクラスに置き換えることと，サーバマシンに変換サーバを導入することで，既存 Web アプリケーションへの変更を最小限に抑えつつ，WebSocket に移行できる．WebSocket に移行することで，Web ブラウザとサーバの通信の応答回数を 1.7 倍にでき，よりリアルタイム性の向上した Web アプリケーションを実現できる．更に HTTP サーバを WebSocket サーバに変更するだけで，コンテンツを一切変更することなく，より WebSocket を活かした高効率な通信を実現できる．

XMLHttpRequest はリクエスト･レスポンス型であるため，本提案方式も WebSocket 上でリクエスト・レスポンス型の通信を行う方式とした．WebSocket の最大の特徴は Web ブラウザからリクエストを必要とせず，サーバから任意のタイミングでプッシュ配信ができることである．XMLHttpRequest を使用する既存の Web アプリケーションは，擬似的にプッシュ配信を実現するために，Web ブラウザからサーバにリクエストを持続的に送信するという設計を採ることが多い．本方式により WebSocket に移行しても，この持続的なリクエストを引き続き行うことになる．プッシュ配信に対応している WebSocket にとって，このリクエストは不必要である．今後，XMLHttpRequest で擬似的に行っていたプッシュ配信処理を，より WebSocket の利点を活かせるように移行する方式を検討する予定である．

参考文献

1)　Ian Hickson: The WebSocket API, W3C Candidate Recommendation (2012)
2)　I. Fette, and A. Melnikov: The WebSocket Protocol, Internet Engineering Task Force, Request for Comments 6455 (2011)
3)　Wordnik: SwaggerSocket: A REST over WebSocket Protocol, https://github.com/wordnik/swaggersocket (2012)
4)　H. Nakajima, M. Isshiki, Y. Takefuji: WebSocket proxy system for mobile devices, Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference