Vol.2019-IOT-44 No.26
2019/3/8

高速Web配信のための コネクション集約とクロスオリジンサーバプッシュの実現

澤田 一樹[1,a)]　北口 善明[1]　山岡 克式[1]

概要：HTTP/2 では，単一接続上での要求多重化やサーバプッシュによりページロード時間（PLT）短縮を図っている．しかし，複数の接続を利用するマルチオリジン構成の Web ページにおいては，十分な PLT 短縮効果が得られない．これに対し，ページ書き換えに基づく接続集約により，多重化効率を向上する手法が提案されているが，スクリプトから動的に要求を送信する Web ページにおいては，この手法では完全な接続集約を行えない．本研究では ServiceWorker 用い，動的な Web ページにおいて完全な接続集約とクロスオリジンサーバプッシュを実現し，従来手法比最大 40%の PLT 短縮を達成した．

キーワード：Web コンテンツ配信，HTTP/2，サーバプッシュ，ServiceWorker

Cross-Origin Server Pushing over Aggregated Connection for Fast Web Distribution

Kazuki Sawada[1,a)]　Yoshiaki Kitaguchi[1]　Katsunori Yamaoka[1]

Abstract: HTTP/2 is aimed to shorten the page load time (PLT) by request multiplexing on a single connection and server pushing. However, the PLT shortening effect will be worsened on multi-origin web pages that need multiple connection. Although the connection aggregation method based on a static page rewriting was proposed to eliminate this issue and improve efficiency, This method is not able to aggregate connection completely for web pages whose script sends subsequent requests dynamically. In this research, we focused on ServiceWorker and utilized it for realizing complete connection aggregation for dynamic web pages and cross-origin server pushing. We achieved up to 40% on PLT shortening compared to the conventional method.

Keywords: Web contents distribution, HTTP/2, Server pushing, ServiceWorker

1. はじめに

Web ページへのアクセスにおけるユーザ満足度は，ページロード時間（PLT）に大きく左右され，ユーザはパフォーマンスの低い Web ページから離れる傾向にある．例えば，大手 E コマースサイトでは PLT が 1 秒大きくなった結果，年間 16 億 USD の損失が発生したとの報告 [1] がある．こうした事例を背景として，Web パフォーマンス改善は重要な課題であると広く認識されている．

Web ページの取得に広く使われているプロトコルである HTTP/1.1 は，1990 年代に開発された単純なプロトコルであり，現代の複雑化した Web ページの取得においては，多くの制約を残している．この制約が，Web パフォーマンスを低下させる大きな要因の一つとなっている．この問題を根本的に解決するために開発された HTTP/2 では，1 つのコネクション上に複数のリクエストを多重化することで HTTP/1.1 の課題を解決しているほか，明示的な要求を受け取る前にコンテンツを送信する機能（サーバプッシュ）が新たに追加された．

しかしながら，現代の Web ページでは，物理的に別のサーバ上に存在するコンテンツに依存するものがほとんど

[1] 東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo, 152–8550, Japan
[a)] sawada@net.ict.e.titech.ac.jp

であり，この場合異なるオリジン[*1]毎に別のコネクションを確立する必要がある．こうした構成では，HTTP/2 の要求多重化やサーバプッシュによるメリットを十分に得られない．この課題に対し，静的解析に基づくページ書き換えでコネクションを集約し，HTTP/2 の特性に即した配信を行う手法 [2] が提案されているが，これは JavaScript から動的に送信される要求を発見できず，集約できない．

本研究では，ブラウザを高度に制御可能である Service-Worker に着目し，これを Web 高速化へ応用する．これによって，[2] では困難であった動的ページに対する完全なコネクション集約の実現，さらには，集約コネクションに基づくクロスオリジンプッシュの実現によって，従来手法が残した課題を解決しながら，HTTP/2 の効率低下問題を解消する手法を提案する．また，提案手法の PLT 削減性能を計測することによって，特に高遅延環境下や高パケットロス環境下における提案手法の有効性を示す．

2. Web 配信技術とその課題

2.1 HTTP/1.1

Web ページの取得に広く使われているプロトコルである HTTP/1.1 では，図 1 に示すように，1 コネクションあたり同時に 1 つのリクエストしか送信できないという制約がある．これに起因して，DB アクセスを伴うリクエストや大容量ファイルへのリクエストなど，処理に時間のかかるリクエストがあると，その待ち時間によって後続のリクエストが遅延する現象が発生し，これは Head-of-Line（HOL）ブロッキングと呼ばれる．基本的に，Web ページは関連する全てのコンテンツの読み込みが完了しない限り完全な表示ができないため，HOL ブロッキングは Web ページの PLT を増大させる大きな要因となっている．

HTTP/1.1 には，Web ページの読み込みを高速化するため，1 つのコネクション上でリクエストの完了を待たずに次のリクエストを送る HTTP パイプラインという仕様が含まれている．しかしながら，HTTP パイプラインでは，サーバは必ず受け取ったリクエストと同じ順番でレスポンスを返さなければならないという仕様上の制限が存在する．このため，小さなコンテンツに対して大量のリクエストを送信する場合に高速化が見込めるものの，遅いレスポンスに起因する HOL ブロッキングの解消には至っていない．

2.2 HTTP/2

HTTP/2 は，HTTP/1.1 との後方互換性を維持しつつ，Web パフォーマンスの向上を図る新たなプロトコルである．HTTP/2 では，コンテンツのオリジンごとに単一の TCP コネクションを確立し，その上に複数のリクエストを多重化する．これは HTTP パイプラインに近いが，HTTP パイプラインではレスポンスの順序がリクエストの順序と同じである必要があったのに対し，HTTP/2 ではレスポンス順序の非同期が許されており，HOL ブロッキング問題を解消している．しかし，昨今の Web ページは複数のサーバにコンテンツを分散配置する複数オリジン構成が一般的であり，この構成ではコンテンツが物理的に別のサーバに配置されている可能性があるため，ブラウザはオリジンの数だけコネクションを確立せねばならず，HTTP/2 のメリットを十分享受できない．

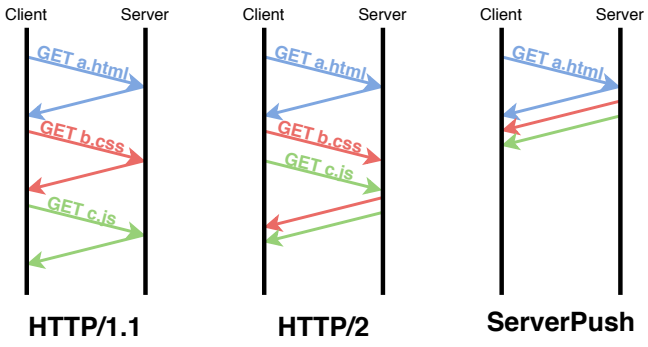

図 1 HTTP の通信フロー
Fig. 1 Communication flow of HTTP

2.2.1 コネクションの再利用

HTTP/2 では，以下の条件を満たす場合，別オリジンへのリクエストを単一コネクション上で送信しても良い．

- それらの FQDN が全て同じ IP アドレスに解決される
- 証明書がそれらの FQDN 全てに対して有効である

この仕様に従えば，複数オリジン構成においても単一のコネクションを利用して通信させることが可能である．

しかし，全ての FQDN に対しての DNS 参照は依然として必要であり，これが PLT 増加の原因となり得る．また，全ての FQDN に対して有効な単一の証明書を用意しなければならないため，証明書管理のコストが高く，サーバ構成の変更が難しくなるなどの課題もある．そもそも，他のサービス事業者など，物理的に別のサーバから提供されるコンテンツに依存するようなページには適用できない．

2.2.2 HTTP/2 サーバプッシュ

通常，Web ページは単一のファイルで完結するものではなく，その依存関係は参照元のファイルを解析して初めて明らかになるものである．したがって，ページを表示するためには少なくとも数回のラウンドトリップが必要となるが，この回数が増えることによって，特に高遅延環境で PLT 劣化が深刻となる．

しかし，コンテンツ提供者側からすると，この依存関係は最初から把握できているはずである．したがって，図 1 に示すように，メインコンテンツへのリクエストがあった時，依存コンテンツへの明示的な要求が来る前にサーバ側から送信してしまうことで，ラウンドトリップ 1 回で全てのコンテンツが取得可能となり，PLT が大きく改善され

[*1] コンテンツ提供元のこと．スキーム，ホスト名，ポート番号によって区別される．

る．この機能は HTTP/2 の仕様に含まれており，サーバプッシュと呼ばれている．

HTTP/2 では，単一のコネクションのみを用いることを原則としており，このサーバプッシュにおいても，それは例外でない．したがって，別のオリジンから提供されるコンテンツをプッシュできない制約がある．また，サーバプッシュではコンテンツの依存関係を予め定義しておく必要があるため，コンテンツに変更があるたびにこの依存関係も修正する必要があり，コンテンツ提供者側の負担が大きくなるという課題も残されている．

管理の手間の問題などから，現状ではサーバプッシュの利用は極めて限定的である．2017 年の調査 [3] によると，1.5 億件の Web サイト中，サーバプッシュを利用している Web サイトは 0.01%以下であり，1 年前と比較しても増加する傾向が見られない．

3. 既存研究

3.1 エッジサーバでのレンダリングキャッシュ [4]

中野らは，2.1 節で述べた HOL ブロッキング問題のほか，性能の低いクライアントマシンでのレンダリング時間に注目し，Web ページのレンダリングをエッジコンピューティングによって行い，この結果をキャッシュするサーバを配信ネットワーク上に配置する手法を提案している．

[4] では，全ての依存コンテンツをエッジサーバ上に集約し，レンダリングまで行った上でキャッシュする．そのレンダリング結果である DOM ツリーや，JavaScript エンジンのメモリ状態など，ブラウザの内部データをシリアライズして送信する．ブラウザは予めインストールされた拡張機能を用いて，シリアライズされたデータを解析し，レンダリング結果を復元する．これによって，Web ページ全体の表示に必要となるリクエストの数が 1 回にまで削減され，特に高遅延環境下においては，大きな PLT 短縮効果が得られると期待されている．

3.2 SPDY アクセラレータによるコネクション集約 [2]

峯木らは，2.2 節で述べたように，近年の Web ページのほとんどが複数のサーバにコンテンツを分散配置する複数オリジン構成であるため，HTTP/2 の効果を十分に得られないばかりか，コネクション確立コストが増大する問題を指摘し，これを解決する手法を提案した．

[2] では，オリジンサーバとブラウザの間に SPDY アクセラレータと呼ばれるサーバを配置し，ここで依存コンテンツへの参照を全て SPDY アクセラレータ宛に書き換えたコンテンツとして配信する．これを受け取ったブラウザは，依存コンテンツへの要求を SPDY アクセラレータへ送信するため，SPDY アクセラレータはリバースプロキシとして動作し，本来のオリジンサーバから取得したコンテンツを返す．これによって，ブラウザと SPDY アクセラレータ間の 1 つのコネクションのみで HTTP/2 を利用できるため，コネクション確立時間の削減が期待される．

4. 本研究の目的

本節では，Web 高速化の基本的アイデア，および既存手法の課題を整理し，本研究の目指すものについて述べる．

4.1 Web パフォーマンス向上の基本アイデア

従来手法は，コネクション集約とリクエスト数削減に基づいて Web 高速化を図ったものであった．本研究においても，従来手法に則り，これら基本アイデアに基づいた PLT 短縮を目的とする．

4.1.1 コネクション集約

単一コネクションのみを用いて通信を行うことを前提に設計された HTTP/2 では，多くのコネクションを用いるよりも，少ないコネクション上で通信を行ったほうが，DNS ルックアップ時間や TLS ハンドシェイク時間などが少なくなるため，結果として PLT が短くなる．コネクション集約は，HTTP/2 の理念に反して多数のオリジンに散らばった Web コンテンツに対しても，その恩恵を得られるようにするためのアイデアである．

4.1.2 リクエスト数削減

リクエスト数が少なくなるということは，ラウンドトリップ数，つまりブラウザとオリジンサーバ間を往復する通信の回数が少なくなることを意味する．これによって，特にラウンドトリップ時間が大きい場合においては，リクエスト数の削減が直接的な PLT の短縮につながる．

4.2 従来手法の課題

4.2.1 導入困難性

2.2.2 節で述べた HTTP/2 サーバプッシュでは，コンテンツ間の依存関係を事前に定義し，コンテンツ変更に併せて依存関係も修正し続ける必要があり，管理の手間が増大するという課題があった．しかし，この課題については，既存の研究では有効な改善策が検討されていない．

3.1 節で述べた既存研究（エッジサーバでのレンダリング）では，拡張機能をブラウザに導入し，ブラウザの挙動を拡張することを前提としている．しかしながら，拡張機能では，ユーザの個人情報にアクセスするような悪意のある拡張機能を作成することも可能であるため，インストールには細心の注意を払う必要があるほか，ユーザ自身で追加のプログラムをブラウザにインストールする必要があるため，Web 高速化という目的においてはユーザから敬遠されがちである．特にライトユーザに対して拡張機能のインストールを案内することは極めて敷居が高く，Web コンテンツ配信高速化のために拡張機能を用いるのは適当でない．ブラウザの内部データは実装ごとに異なるものであり，これをキャッシュするため，全てのブラウザとそのバージョ

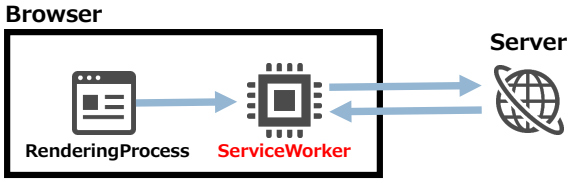

図 2　ServiceWorker によるリクエストへの割り込み
Fig. 2　Request interruption by ServiceWorker

ンごとにレンダリングが必要である．また，一部のコンテンツのみ変更された場合であっても，全体をレンダリングし直す必要があるなど，エッジサーバの負荷増大が懸念される．さらに，環境依存のキャッシュデータ互換性の問題など，実際の運用には多くの課題が残されている．

本研究では，これら問題を解消することが目的である．ユーザに追加の操作を要求せず，かつサービス提供者側で依存関係を管理する必要性のない，透過的に利用可能な高速 Web 配信手法を目指す．

4.2.2　動的ページへの適用困難性

3.2 節で述べた既存研究（SPDY アクセラレータ）では，静的解析に基づいた参照の発見とその書き換えに基づいたコネクション集約を行っていた．しかしながら，静的解析ではスクリプトから動的に送信される参照を発見することは極めて難しく，集約漏れが発生してしまう．さらに，参照先の書き換えは，本来想定されたスクリプトの動作から逸脱させる可能性をはらんでいる．

本研究では，動的ページでも漏れのない完全な集約を実現すること，およびページの動作を破壊する危険性を排除することを目的としている．

5.　提案手法

本手法では，図 2 に示すような，ServiceWorker によるリクエストへの割り込み機能に着目し，これを Web 高速化に応用する．ServiceWorker とは，Web ページとは別のバックグラウンドで動くプロセスを定義し，ブラウザの挙動を変更・拡張する仕組みのことである．拡張機能とは異なり ServiceWorker をインストールした単一オリジンの Web ページ上にしか干渉できないという制約があるものの，Web ページへのアクセスがあった際にユーザの操作を必要とせず，バックグラウンドでインストールすることが可能である．

加えて，本手法では CDN エッジサーバと ServiceWorker を協調動作させることにより，高度な配送経路制御を行う．CDN とは，エッジサーバと呼ばれる利用者の近くに設置されたサーバがリバースプロキシとして動作し，遠い場所にあるオリジンサーバからオリジナルコンテンツを取得してキャッシュすることで PLT を短縮する仕組みのことである．また，CDN はオリジンサーバの負荷分散のために用いられることも多い．CDN は通常リバースプロキシとして動作するが，本手法では ServiceWorker からの命令を受け取ってフォワードプロキシとしても動作する．

さらに，本手法の特徴である，全てのリクエストが単一のエッジサーバを経由する環境を活用することを提案する．エッジサーバが受け取るリクエストを監視することによって，複数オリジンにまたがるコンテンツの依存関係を動的に解決し，サーバプッシュするコンテンツを自動的に選択するクロスオリジンプッシュを実現する．

5.1　リクエストのカプセル化によるコネクション集約

本手法では，ServiceWorker を用いてブラウザが送信する直前のリクエストをカプセル化し，エッジサーバを経由して送ることで，単一のコネクション上でリクエストを完結させる．初回リクエストは CDN エッジサーバに誘導されるようにセットアップされているものとする．本手法の動作フローを以下に示す．

- ブラウザは通常の CDN と同様の方法によって，Web ページをエッジサーバに要求する《図 3 (1)》.
- エッジサーバはこの要求を受け取ると，本来のコンテンツの代わりに ServiceWorker をブラウザにインストールする特殊なコードを送信する《図 3 (2)》.
- インストールが完了《図 3 (3)》すると，ブラウザは再度ページを開くためにリクエストを送信しようとする《図 3 (4)》．このとき，ServiceWorker はリクエストをフックし，元のリクエストをカプセル化したエッジサーバ宛のリクエストを作成し，送信する《図 3 (5)》.
- カプセル化されたリクエストを受け取ったエッジサーバは，元のリクエストを取り出してオリジンサーバに送信する．オリジンサーバからコンテンツを受け取ると，これをそのまま ServiceWorker へ送信する《図 3 (6)》．または，自身にそのコンテンツがキャッシュされているならば，これを ServiceWorker へ送信する．
- エッジサーバからのレスポンスを受け取った ServiceWorker は，それがあたかも本来のオリジンサーバから返ってきたものであるかのようにブラウザのレンダリングプロセスに渡す．レンダリングプロセスはそのコンテンツの解析を行い，依存コンテンツへの要求を送信しようとする．このリクエストも同様に ServiceWorker によってフックされ，カプセル化されてエッジサーバに送信される．

初回のみエッジサーバにキャッシュが存在しないため，エッジサーバは通常のブラウザと同様に振る舞い，複数のオリジンサーバへコネクションを確立する必要がある．そのため，ブラウザとエッジサーバ間でコネクション集約を行っても PLT の短縮効果はない．同じ Web サイトのコンテンツは，同じオリジンサーバから提供されるコンテンツに依存する可能性が高いことを考えると，エッジサーバとオリジンサーバ間で一度確立したコネクションを長時間維

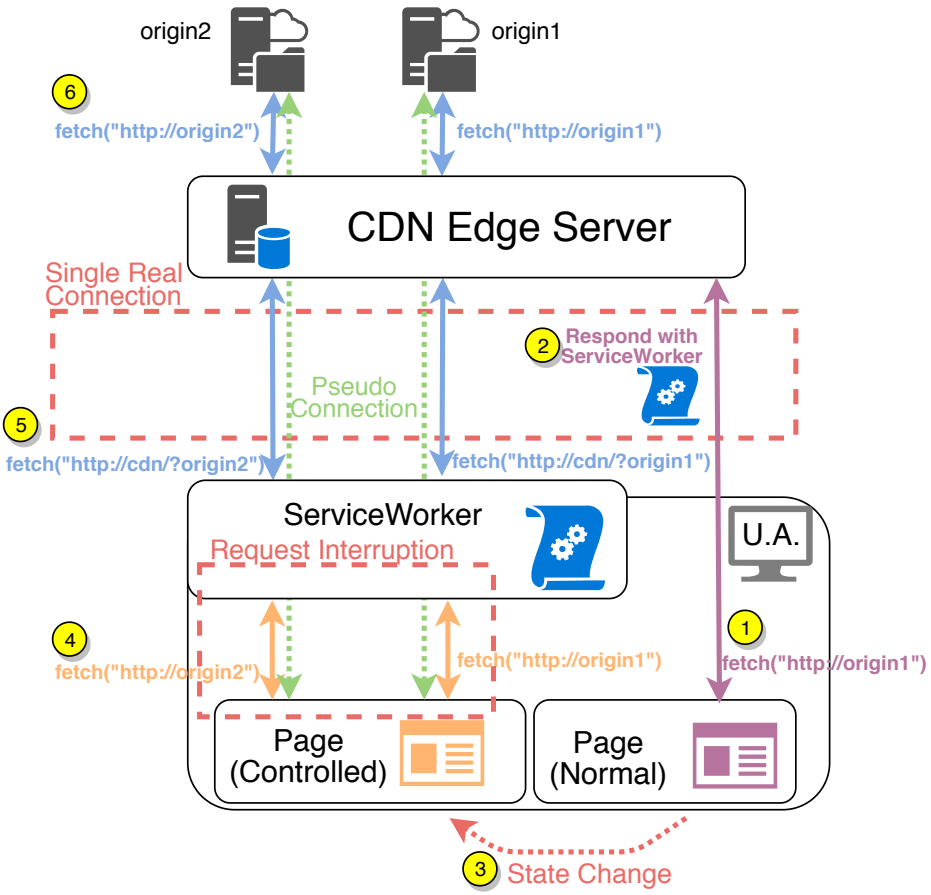

図 3　提案手法の動作の流れ
Fig. 3　Flow of proposed method

持し続けることで，初回リクエストに対しても PLT の短縮可能性がある．なお，ServiceWorker は一度インストールされると同一オリジン内のページを全てコントロールすることが可能で，ブラウザを再起動しても再び同一の ServiceWorker が起動するので，ServiceWorker をインストールするコードの送信はその Web サイトのいずれかのページを初めて開く時だけで良く，それ以降の同一サイト内での遷移時には不要である．

以上によって，依存コンテンツが別オリジンから提供されるものであっても，ServiceWorker によって単一コネクション上で全てエッジサーバへ送信されるようになった．これによって，DNS ルックアップやコネクション確立のオーバヘッドが削減され，PLT の改善が期待される．

5.2　クロスオリジンプッシュ

本手法では，全てのレスポンスがエッジサーバを経由してブラウザへ送信されるため，コンテンツ間の依存関係が分かれば，オリジンにかかわらず全てのコンテンツをサーバプッシュ可能である．また，全てのリクエストをエッジサーバで中継するため，この順序情報から完全な依存関係解決が可能である．以下に，依存関係の解析手法を述べる．

あるブラウザが，Web ページの起点となるコンテンツを要求した後に続いて要求したコンテンツは，最初に要求したコンテンツが依存しているものである可能性が高い．この原則に基づけば，ブラウザが送信してくるリクエストの順序を監視することによって，その依存関係を解決することが可能である．

具体的には，起点となる要求の後に，同一のブラウザから到着したリクエストを 1 つのセッションとみなし，起点のリクエストに対応するコンテンツは，セッションに含まれるリクエストに対応するコンテンツに依存するものとみなす．この動的解析によって，従来の静的解析では困難であった，スクリプトから動的に要求が送信されるコンテンツの依存関係についても解決が可能である．こうして解決した依存関係を用いて，リクエストがあった際にプッシュするコンテンツを選択する．

この方法では，図 4 に示すように本来は間接依存であったコンテンツについても，直接依存として解決されてしまう．しかしながら，起点リクエスト時に依存コンテンツを全てプッシュすることを目的としているため，この形の依存関係解決でも十分である．

以下に，本手法のサーバプッシュ戦略を示す．

- エッジサーバは，あるコンテンツに対して要求があった時，そのコンテンツに関する依存関係が解決済みか

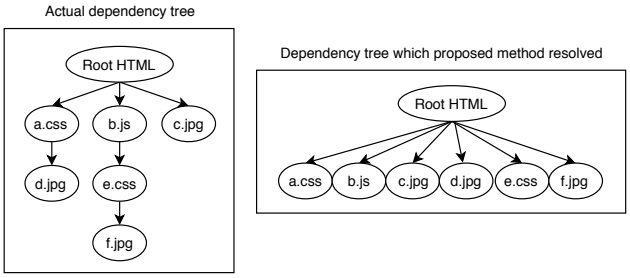

図 4　依存関係解決
Fig. 4　Dependency resolution

を確認する．解決済みであれば，そのコンテンツが依存するコンテンツを全てプッシュする．

- 未解決であれば，プッシュは行わない．このとき，レスポンスを送信したブラウザが次に送ってくるリクエストを記録する．
- 一定時間が経過した後，リクエストのログを解析してセッションを同定し依存関係を解決する．

以上によって，スクリプトから要求が送信される動的ページを含む任意のページにおいて，依存関係解決を行うことが可能となり，これに基づいたクロスオリジンプッシュが実現可能となった．クロスオリジンプッシュにより，従来プッシュできなかったオリジンが異なるコンテンツについてもプッシュを可能とした．これによって，プッシュされるコンテンツ数が増加し，HTTP/2 サーバプッシュのリクエスト数削減効果が大きく向上することが期待される．

5.3　セッション同定

ユーザは通常，ブラウジング中に何度もページ間を遷移し，複数のページを同時に開く可能性がある．したがって，起点リクエストの判定とセッションの同定は大変難しい．Web サーバログ解析によるセッション同定は多数検討されている [5]．本手法で観測するリクエストは，一般の Web サーバログとほぼ同等のものであるため，それらログ解析手法に関する研究成果が応用可能である．

なお，セッション同定が誤っていたとしても，ページの閲覧自体に影響はない．必要ないコンテンツをプッシュしても読み捨てられるのみであり，プッシュされなかったコンテンツについてはブラウザが再要求する．しかしながら，誤ったサーバプッシュは帯域の浪費や PLT 短縮効果の低下を招くため，高精度のセッション同定が理想である．

5.4　従来手法との比較

この手法はユーザと CDN エッジサーバ間で完結するものであるため，コンテンツに手を加える必要がない．さらに，ServiceWorker を用いているため，ユーザはこの配信手法の存在について意識する必要がない．したがって，4.2.1 節で述べた導入困難性を解決しており，従来手法と比較して，ユーザとサービス提供者の双方にとって，利用にかかる手間が小さくなっている．

ServiceWorker がリクエストに割り込むことで，動的な参照を完全に発見可能であり，動的ページにおいてもコネクション集約が可能となった．さらに，リクエストのカプセル化によって参照の書き換えも不要であるため，ページの振る舞いを破壊する可能性を排除している．したがって，4.2.2 節で述べた動的ページへの適用困難性を解決しており，従来手法が苦手とする動的ページにおいては，その Web パフォーマンス改善効果が高くなると予想される．

6.　評価

本節では，提案手法を実装し計測・評価を行った結果について述べる．通常の配信方法と，従来手法（SPDY アクセラレータ）を用いる配信，および提案手法とを比較した．

6.1　計測環境

2019 年 1 月における標準的な Web ページ [6] に近い構成のページ（5 オリジン，リクエスト数 75，容量 1.5 メガバイト）を用意し，このページに対する PLT を計測した．このページでは，メイン HTML から CSS を参照し，その CSS からさらに画像を参照するような，一般的な 3 段参照となっている．また，一部の依存コンテンツを JavaScript から動的に読み込む構成としている．

主コンテンツの要求開始から，全ての依存コンテンツの読み込みが完了するまでを PLT とした．読み込み完了のタイミングはブラウザの Resource Timing API を利用して検知している．Chromium 70 を DevTools API で自動制御し，先述した Web ページの PLT を 100 回計測した上で，上位 5%と下位 5%を除いた平均の値を用いた．

モバイルネットワークでの Web の利用増加を考慮し，netem[*2]を利用して通過するパケットを意図的に遅延・損失させるためのルータを図 5 に示すようにクライアントとエッジサーバ・オリジンサーバ間に設置した．この環境では，HTTP の通信のみではなく，DNS など全ての通信が遅延・損失する．オリジンサーバ，エッジサーバ，ルータ，クライアントは全て仮想マシンであり，同一の仮想ネットワーク上に存在し，全ての仮想インターフェイスのリンク帯域は 1Gbps である．このため，通常の CDN エッジサーバが持つ近距離からの配信による PLT 短縮効果は存在せず，純粋な提案配信手法の評価となっている．

6.2　HTTP/3 を用いた計測

HTTP/2 では，原則として 1 つの TCP コネクションしか使用しないため，TCP の通信品質が劣化すると，ほぼ全てのリクエストがその影響を受けることになる．HTTP/2

[*2] Linux 上で動作するネットワークエミュレーションプログラム．

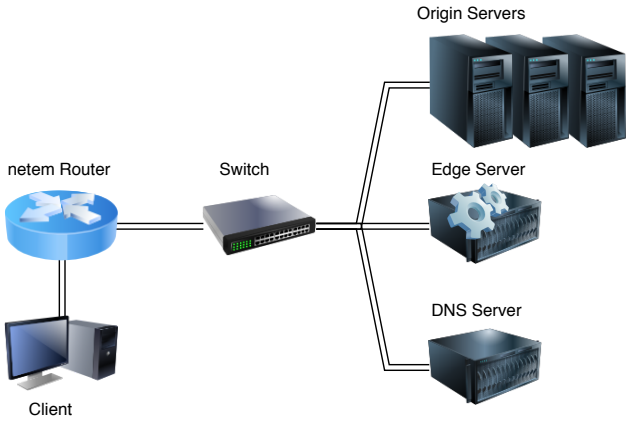

図 5　計測環境
Fig. 5　Measure environment

では TCP の通信品質劣化の影響を大きく受け，場合によっては HTTP/1.1 より通信品質が悪くなることが指摘されている [7]．このため，高パケットロス環境下において，コネクション集約による PLT 短縮を図る従来手法および提案手法では，HTTP/2 による通常配信よりも更に通信速度が劣化することが予想される．

本研究では，高パケットロス環境下における従来手法と提案手法の評価も行う．さらに，トランスポート層に QUIC を用いる HTTP/3[*3] 上での性能についても比較計測を行う．従来手法や提案手法は，HTTP/2 上での動作を前提としたものであったが，HTTP/3 には HTTP/2 と同等の機能が含まれるため，HTTP/3 上でも動作可能である．

この QUIC とは，UDP 上で TCP に相当する信頼性や輻輳制御と，TLS に相当するセキュリティを実現するトランスポート層プロトコルである．QUIC の開発の動機は，先述した TCP 通信品質劣化による HTTP/2 の性能悪化問題を根本的に解決することにある．

未だ実験段階であることや，その仕様の複雑さから，現状では HTTP/3 に対応した Web サーバは極めて少数で，その殆どが実験的な実装であるため，プロダクション環境での利用例は Google を除いてほぼ存在しない．HTTP/3 に対応する Web ブラウザも，Chrome などごく一部のみである．しかし，その名称からも分かるように，今後は急速に普及が進んでいくことが予想されている．

6.3　計測結果

6.3.1　遅延に対する PLT の変化

この計測では, プッシュ効率の向上を示すため，従来手法についても 5.2 節で述べた自動クロスオリジンプッシュを適用している．パケットロス率を 0%に固定し，遅延を変化させた場合の計測結果を図 6 に示す．

図 6 を見ると，提案手法は全ての遅延環境下において良好な性能を記録している．通常配信では，特に高遅延環境下で，通信確立コストが PLT に与える影響が顕著に現れている．また，動的なコンテンツ要求に対する接続を集約できない従来手法に対し，提案手法では，これを完全に集約可能であることから，PLT が 22〜32%減少している．

[*3] QUIC 上で動く HTTP で，HTTP/1.1 や HTTP/2 と異なる．

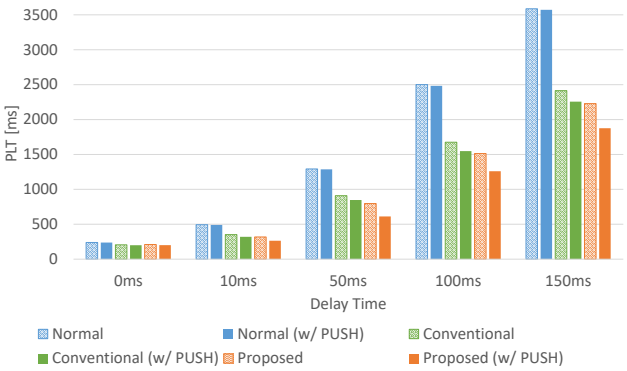

図 6　遅延時間に対する PLT
Fig. 6　Delay time vs. PLT

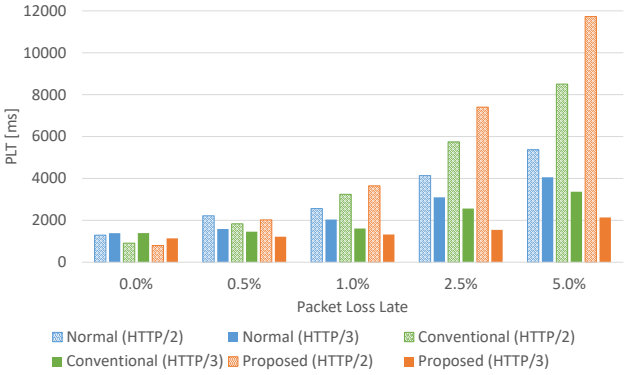

図 7　パケットロス率に対する PLT
Fig. 7　Packet loss rate vs. PLT

また，提案手法ではサーバプッシュの PLT 短縮効果が大きいことが分かる．提案手法の一部である，クロスオリジンプッシュを通常配信および従来手法にも適用した場合，通常配信では 1%以下，従来手法では 6〜9%減であるが，提案手法では 15〜23%の PLT 短縮効果があった．これは，完全なコネクション集約によって，従来手法より多くのコンテンツのプッシュが可能となるためであり，提案手法はサーバプッシュとの親和性が高いことが分かる．

6.3.2　パケットロス率に対する PLT の変化

この計測では，パケットロス耐性のあるプロトコルである QUIC と，その上で動作する HTTP/3 をそれぞれの配信方法に適用した場合についても計測を行った．遅延を 50ms に固定し，パケットロス率を変化させた場合の計測結果を図 7 に示す．

図 7 を見ると，HTTP/2 を用いた場合，高パケットロス環境下において，従来手法や提案手法における PLT は通常配信よりはるかに大きくなっている．これは，単一の TCP コネクションのみを用いるため，接続環境が悪化しトラン

スポート層における HOL ブロッキングが発生したことで，全てのリクエストがその影響を受けるためである．特にコネクション集約度が高い提案手法においては，従来手法と比較しても性能劣化の度合いが極めて顕著である．

対して，HTTP/3 を用いた計測では，HTTP/2 を用いた場合と比較して性能劣化を小さく抑えられている．パケットロス率が 0%から 1%になったとき，通常の HTTP/2 を用いる提案手法では PLT が 358%増加しているが，HTTP/3 では 16%増にとどまっている．これは，HTTP/2 がトランスポート層で用いている TCP と比べて，HTTP/3 がトランスポート層で用いる QUIC では，HOL ブロッキングが発生しないように設計されているなど，パケットロスに対する耐性が高いためである．また，HTTP/3 上でも提案手法は従来手法と比較して良好な性能を示しており，最大で 40%の PLT を短縮している．

以上から，提案手法は QUIC との相性が良く，QUIC を用いることで高パケットロス環境においても現実的な性能を示すことが分かる．なお，パケットロス率が 0%のとき HTTP/3 上では PLT が若干増加しているが，これは良好な環境下では TCP より性能が劣化する QUIC の特性によるものである [8].

7. 今後の課題

7.1 Spped Index とプッシュ戦略

本提案手法では，全ての関連コンテンツを一括でサーバプッシュすることによって，PLT を短縮した．しかしながら，この最も単純なプッシュ戦略は，QoE 向上において必ずしも最適ではない．

Speed Index は，ユーザがページを開いてはじめに目にする部分の描画スピードを測るメトリクスであり，これは PLT よりも現実的な QoE の指標になり得る．ページの末尾の描画に必要なコンテンツを大量にプッシュするよりも，ユーザがはじめに目にする部分のみをプッシュすることで，QoE をより改善できる．

今後は，Speed Index をメトリクスとして用いた本提案手法の評価や，その上で各コンテンツの性質に応じたプッシュ戦略を用いることで，実際の QoE と関わりの強い Speed Index においても高い性能を発揮できるような改良方法について検討を行う．

7.2 機密性と完全性

本提案手法では，TLS 通信をエッジサーバ上で終端するため，機密性と完全性は保証されない．これは通常の CDN でも同様であり，本提案手法のセキュリティリスクは通常の CDN と同程度である．いずれの場合でも，ユーザとサービス提供者の双方が，CDN 事業者を信頼する必要性がある．ただし，本手法のように ServiceWorker で制御されたコンテンツについては，CDN エッジサーバで TLS が終端されているのか，オリジンサーバまで TLS で接続されているのかをブラウザ上では判別できず[*4]，ユーザがそのコンテンツの安全性を判断する上で大きな障壁となり得る．現在ではこの挙動は主要ブラウザで共通であるが，潜在的な脅威となりうるため，今後のアップデートで変更がなされる可能性がある．

今後は，コンテンツの機密性と完全性を提案手法上で保証可能とする拡張について検討する．

8. おわりに

本研究では，ServiceWorker を Web パフォーマンス改善に応用する手法を提案した．ブラウザへの拡張機能導入を必要とせず，サービス提供者が膨大なコンテンツに手を加える必要なく，従来手法では困難であった動的 Web ページに対しても，完全なコネクション集約とクロスオリジンプッシュの実現が可能であることを示した．これらの実現によって，従来手法比最大 40%の PLT 短縮効果が得られた．今後は，より良いプッシュ戦略の模索や，コンテンツの機密性・完全性を保証する手法について検討する．

参考文献

[1] V. Green, “Impact of slow page load time on website performance,” https://medium.com/@vikigreen/impact-of-slow-page-load-time-on-website-performance-40d5c9ce568a, （2019 年 2 月 4 日 閲覧）.
[2] G. Mineki, S. Uemura, and T. Hasegawa, “Spdy accelerator for improving web access speed,” Proceedings of the International Conference on Advanced Communication Technology (ICACT) 2013, pp.540–544, Jan. 2013.
[3] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld, “How HTTP/2 pushes the web: an empirical study of HTTP/2 server push,” Proceedings of the IFIP Networking Conference (IFIP Networking) 2017, pp.1–9, June 2017.
[4] Y. Nakano, N. Kamiyama, K. Shiomoto, G. Hasegawa, M. Murata, and H. Miyahara, “Web performance acceleration by caching rendering results,” Proceedings of the Asia-Pacific Network Operations and Management Symposium (APNOMS) 2015, pp.244–249, Aug. 2015.
[5] A. Sawabe, H. Yoshida, and K. Nogami, “Log analysis in a HTTP proxy server for accurately estimating web QoE,” Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC) 2018, pp.1–7, Jan. 2018.
[6] I. Grigorik, P. Meenan, R. Viscomi, and P. Calvano, “HTTP Archive,” https://httparchive.org/, （2019 年 2 月 4 日 閲覧）.
[7] N. Oda and S. Yamaguchi, “HTTP/2 performance evaluation with latency and packet losses,” Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC) 2018, pp.1–2, Jan. 2018.
[8] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., “The QUIC transport protocol: Design and Internet-scale deployment,” Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM) 2017, pp.183–196, Aug. 2017.

[*4] 開発者ツールなどを利用することで確認可能である．