Vol.2013-CSEC-61 No.21
Vol.2013-IOT-21 No.21
2013/5/10

WebSocketによる終了検知を行うOpengateの開発

大谷 誠[1,a)]　江藤 博文[1]　渡辺 健次[2]　只木 進一[1]　渡辺 義明[3]

概要：佐賀大学では，利用者端末や公開端末からのネットワーク利用を認証・記録する Opengate を開発・公開し，学内において運用を行ってきた．Opengate は，HTTP Keep-Alive と遅延応答によってコネクションを維持し，このコネクションを用いて端末の利用を監視していた．しかし，遅延応答を Web ブラウザが誤認識し，コネクションを予期せぬタイミングで切ってしまうという場合があった．そこで，終了検知手法として WebSocket を用いる新たな Opengate を開発した．

キーワード：Opengate, ネットワーク利用者認証, WebSocket

Development of Opengate Capable of Detecting Usage Termination by WebSocket

Makoto Otani[1,a)]　Hirofumi Eto[1]　Kenzi Watanabe[2]　Shin-ichi Tadaki[1]　Yoshiaki Watanabe[3]

Abstract: We have developed and distributed a network user authentication system Opengate. It has been operated in Saga University. Opengate establishes a connection with a terminal using “HTTP Keep-Alive” and “Delayed response”. Opengate monitors network use of the terminal using the connection. But, current Opengate has a problem that the connection for monitoring is arbitrarily disconnected by Web browser. So, we developed new Opengate which used WebSocket for detection of termination.

Keywords: Opengate, network user authentication, WebSocket

1. はじめに

佐賀大学では，ネットワーク利用の認証・記録を行う Opengate を開発・公開し，学内で運用を行っている [1]. Opengate は，Web ブラウザを用いてネットワーク利用の認証を行う．認証終了後，Web ブラウザ上で JavaScript を実行し，Opengate の監視プロセスとクライアント間にコネクションを確立する．Web ブラウザの終了等によってコネクションが切断されると，クライアントがネットワークの利用を終了したと判断しファイアウォールを閉鎖し利用ログを出力する．

コネクションの維持には，HTTP Keep-Alive と遅延応答を用いて行っている．しかしこの方法において，Web ブラウザが遅延応答をサーバ側のエラーだと誤認識しコネクションが勝手に切断してしまう場合があった．

そこで WebSocket コネクションを用いて，ネットワークの利用終了を検知する新たな Opengate の開発を行った．WebSocket コネクションの実装には，サーバサイドの JavaScript 実行環境である Node.js を用いた．本稿では，Opengate の概要を述べた後，WebSocket コネクションを用いてネットワークの利用終了を検知するこの新たな Opengate の概要について述べる．

2. Opengate

この章では，Opengate の概要について述べる．

[1] 佐賀大学総合情報基盤センター
Computer and Network Center, Saga University
[2] 広島大学大学院教育学研究科
Graduate School of Education, Hiroshima University
[3] 佐賀大学大学院工学系研究科
Graduate School of Science and Engineering, Saga University
[a)] otani@cc.saga-u.ac.jp

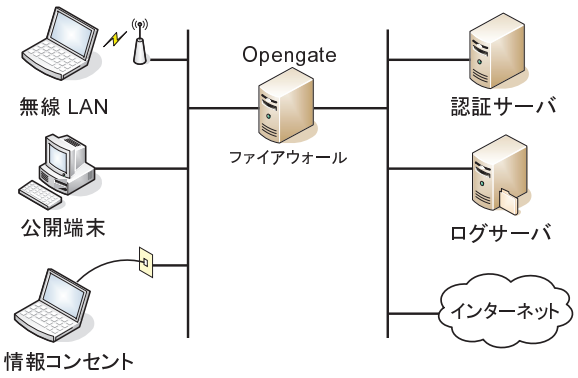

図 1 Opengate のシステム構成例
Fig. 1 Sample configuration of Opengate system

2.1 概要

佐賀大学では Opengate と呼ばれる Web ブラウザを使った Captive Portal 型のネットワーク利用者認証システムを開発・公開している．また，学内において全学規模で運用を行っている．この Opengate は，認証に Web ブラウザを用いることにより，公開端末や個人所有のノート PC などの様々な種類の端末利用者に，共通の認証基盤を提供する．認証には既設の LDAP や RADIUS サーバなどの認証サーバを利用することができ，IPv6 のネットワーク [2] や，Shibboleth を用いたシングルサインオン認証 [3] にも対応している．

Opengate は，認証を行った Web ブラウザのウィンドウの閉鎖による監視用のコネクションの切断をネットワーク利用終了と判断し，ファイアウォールを閉鎖する．また Opengate では，認証で得られた利用者の情報，端末情報，利用開始・終了時刻を記録する．佐賀大学では，現在は学内のほぼ全ての教室の情報コンセントおよび無線 LAN で Opengate の使用が可能となっている．会議室や研究室にもサービスを提供し，学生だけでなく教員も利用している．

Opengate のシステム構成例を**図 1** に示す．

2.2 HTTP による利用終了の検知とその問題点

Opengate は，Web ブラウザを用いてネットワーク利用の認証を行う．Web ブラウザ上で JavaScript を実行し，Opengate の監視プロセスと HTTP による非同期通信 (XMLHttpRequest) を行い，サーバとクライアント間にコネクションを確立する．Web ブラウザの終了等によってコネクションが切断されると，利用終了と判断する．

通常，HTTP 通信は一度のデータ送受信で切断される一過性の TCP コネクションを用いるため，その TCP コネクションの終了をもって利用者がネットワーク利用を終了したと判断することはできない．しかし HTTP/1.1 において，1 回の TCP 接続で複数の HTTP リクエストを処理することが可能な HTTP Keep-Alive 機能が追加され，多くの Web ブラウザで利用できるため，Opengate はこの機能を利用している．

ただし HTTP Keep-Alive による TCP コネクションも，一定時間以上データのやり取りを行わなわなければ，タイムアウトとなり切断されてしまう．このタイムアウトまでの時間は，Web ブラウザのバージョンや種類によって大きく異なる．また，Web ブラウザによっては，利用状態に関係なく一定時間 (30 秒など) で切断するものもある．Web ブラウザによって，TCP コネクションを維持可能な時間が異なるため，HTTP Keep-Alive のみでは，TCP コネクションの長時間維持が困難である．

そこで，この問題に対応するため HTTP の応答を遅延することで，HTTP Keep-Alive による維持時間を短くしたまま，通信の繰り返しの周期を延ばす手法を Opengate では用いた．この手法では，Web ブラウザと Opengate 間で定期的に hello メッセージの交換を行う．Web ブラウザからの hello メッセージに対して監視プロセスが一定時間遅延させて返答する．この際の応答 (遅延応答) までの時間は，HTTP Keep-Alive ではなく，Web ブラウザの標準的な機能によって TCP コネクションの維持が可能である．Web ブラウザが応答を受けて, 再度 hello メッセージを送信する短時間の間は，HTTP Keep-Alive による TCP コネクションを維持している [4].

しかしながら，Web ブラウザによってはこの遅延応答を誤認識し，強制的に Web ブラウザがコネクションを切断してしまう場合が確認できた．そこで，HTTP Keep-Alive や遅延応答によって発生する問題を解決するために，WebSocket を用いた新たな Opengate の開発を行った．

3. WebSocket を用いた新たな Opengate

この章では，WebSocket を用いた新たな Opengate について述べる．

3.1 WebSocket と Node.js

WebSocket とは，Web サーバと Web ブラウザ間で双方向通信可能なプロトコルである．これまで Opengate で使用していた XMLHttpRequest の欠点を解決する技術として開発されており，Ajax や Comet 等に変わる技術として注目されている．当初 HTML5 の仕様の一部だったが，後に HTML5 から切り離され，現在は単独のプロトコルとして規格策定が進められている [5], [6].

従来使用している HTTP 通信は，クライアントがリクエストをサーバに送り，そのリクエストに対してサーバがレスポンスを返すという形だった．WebSocket はそれとは異なり，1 度 WebSocket コネクションを張るとクライアントとサーバはお互い任意のタイミングでデータの送受信を行うことができ，必要な通信全てをそのコネクション上で行う．任意のタイミングで送受信が行えるので，リアルタイムな通信が可能である．1 つのコネクションでデータ

の送受信を行えるため，データ送受信の度に新たにコネクションを行う必要がなくサーバの負担が減るというメリットがある．また，WebSocket で行う通信には独自のプロトコルが用いられており，ヘッダが小さく軽量なプロトコルとなっている．

WebSocket を扱うことのできるサーバサイドの実行環境は多く存在するが，新たな Opengate においては，“Node.js” を利用した．Node.js とは，サーバサイドでの JavaScript の実行環境の 1 つである．JavaScript は主に Web ブラウザ上で動作する言語だが，Node.js はその JavaScript をサーバ上で実行することができる [7], [8]．JavaScript エンジンとして Google 社製 V8 エンジンを搭載している．特徴として，シングルスレッドで非同期処理を行うことである．また，そのためのイベントループとノンブロッキング I/O という仕組みがある．

イベントループとは，多数アクセスを処理するためのアーキテクチャである．イベントループは，イベントキューに実行する処理を 1 度登録し，順に実行していく．実行が完了した処理の結果は，イベントキューにコールバック関数を登録することで受け取ることができる．シングルスレッドで登録された順に実行していくため，途中に処理待ちが発生するようなブロックする処理がある場合，その処理が完了した後に次の処理が実行される．このようなブロックを無くし効率よく処理を行うため，ノンブロッキング I/O を使う．

WebSocket コネクションの実装には，Node.js 用のモジュールである Socket.IO を用いた．Socket.IO を用いることで簡単に WebSocket 通信が実現できる．また，WebSocket 非対応の Web ブラウザに対する互換性もあるため，多くの Web ブラウザに対応しやすい [9].

3.2 システム構成

従来の Opengate は認証後，Web ブラウザに許可ページを送る．許可ページは Opengate の CGI が立ち上げた監視プロセスへ監視ページの要求を行う．そして送られてきた監視ページが監視プロセスとコネクションを張り，監視プロセスはネットワークの利用を監視する．Web ブラウザの終了等によってコネクションが切断されると，監視プロセスがファイアウォールを閉鎖し利用終了のログを記録する (**図 2**).

しかし，今回開発したシステムでは Opengate の認証後，Web ブラウザに許可ページを送るが監視プロセスは立ち上げない．監視プロセスの代わりとなるプログラム server.js を予め Node.js 上で実行し待機しておく．許可ページは server.js へ監視ページを要求し，送られてきた監視ページは server.js と WebSocket コネクションによる通信を行う．WebSocket コネクションが切断されると監視プロセスと同様にファイアウォールの閉鎖と利用終了のログを記録する

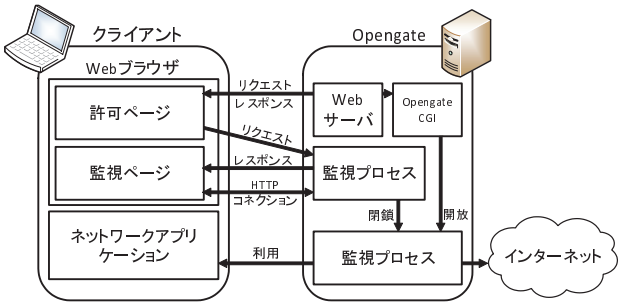

図 2 従来の Opengate のシステムの構成
Fig. 2 System architecture of old Opengate

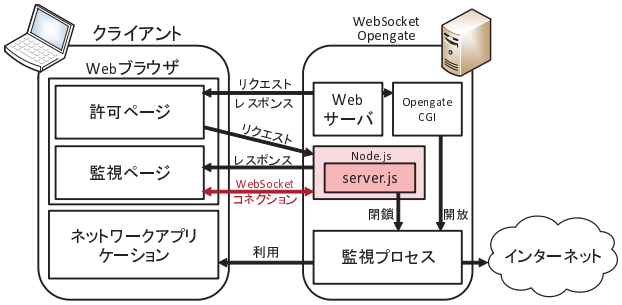

図 3 新しい Opengate のシステムの構成
Fig. 3 System architecture of new Opengate

(**図 3**).

このようにシステム全体ではなく監視プロセス部分の変更としてシステムを構築した．よって Opengate からの移行が容易である．また，安定した移行が行えるように新たな Opengate は Node.js で監視を行う方式と，従来の監視プロセスで行う方式を切り替え可能となるよう構築している．

3.3 システムの動作

以下に，WebSocket を用いた新たな Opengate の動作の流れを示す．

(1) Opengate は，従来の監視プロセスの代わりとなるプログラム server.js をあらかじめ Node.js 上で実行し待機する
(2) 利用者は公開端末やノート PC 等で任意の Web サイトへアクセスする
(3) Opengate は，この通信を横取り，ユーザ ID とパスワードを要求する認証ページを返す (Shibboleth 認証の場合は，IdP に転送する)
(4) 利用者が入力した認証情報を元に，認証サーバに問い合わせを行い，認証に成功すれば，当該端末に対してファイアウォールを開放するとともに，利用開始の情報をデータベースおよびログに記録する
(5) OpengateCGI は許可ページを Web ブラウザに送信する
(6) 許可ページ内において JavaScript を実行し，Node.js による server.js に対して監視ページ (**図 4**) を要求する

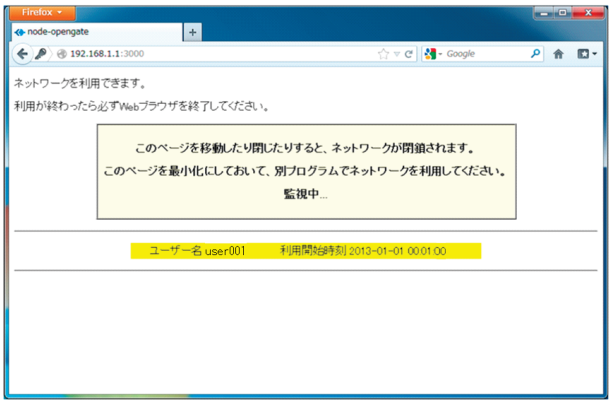

図 4 ネットワーク利用監視ページ
Fig. 4 Network usage monitor page

(7) server.js は，監視ページを Web ブラウザに送信する
(8) 監視ページと server.js との間で WebSocket コネクションを確立し，利用者のネットワーク利用を監視する
(9) WebSocket の切断を検知するとファイアウォールのルールを削除し，ネットワークを閉鎖する
(10)利用終了の情報をデータベースとログに記録する

利用者から見た基本的な利用動作は，従来の Opengate と同様である．そのため利用者は，新たなネットワーク利用の監視を意識せずに，従来通りネットワークの利用ができる．

3.4 WebSocket コネクションと利用者情報との紐付け

従来の Opengate のネットワーク利用の監視には，1 つのクライアントに対して 1 つの監視プロセスを立ち上げ，それを用いて行っている．Node.js で監視を行う方式では，複数のクライアントに対して 1 つのプログラムが監視を行う．従来の方式では，複数のクライアントを相手としないのでコネクションを区別し，判別する必要はない．しかし Node.js で行う場合は，WebSocket コネクションとクライアントを紐付けて区別できるようにする必要がある．そこで新たな Opengate では，以下の方法で，WebSocket コネクションと利用者情報との紐付けを行った．

サーバとクライアント間で WebSocket コネクションを確立する時，ハンドシェイクを行う必要がある．Socket.IO ではハンドシェイク時に認証を挟むことができる．ハンドシェイク時のハンドシェイクリクエストデータからハンドシェイクデータオブジェクトを生成する．このハンドシェイクデータオブジェクトは，Socket.IO による認証後も扱えるので，データを格納し WebSocket コネクション中の処理に格納したデータを扱うことができる．

ハンドシェイクデータオブジェクトはリクエストヘッダの情報を保持しているので，ヘッダから Cookie の取得を行うことができる．Cookie の値を使用しデータベースからユーザ ID や IP アドレス，開放したファイアウォールのルール番号等の利用者情報を取得しハンドシェイクデータオブジェクトに格納する．格納したデータは WebSocket コネクション中の処理に扱うことができるので，コネクションからクライアントを判別する時，格納されたデータから判別ができる．

このようにコネクションとクライアントを紐付けを行う．Opengate の場合は，WebSocket コネクション切断後のログの記録を行う時，格納されたユーザ ID や IP アドレスを使用することで誰がネットワークの利用を終了したのかを記録する．

3.5 Node.js によるファイアウォール (ipfw) の制御

利用終了時のファイアウォールの閉鎖において，ファイアウォールの制御に ipfw を用いているが，この際に root 権限が必要となる．

しかしセキュリティ上，プログラムを常時 root で動かすのは好ましくない．従来の Opengate は ipfw を実行する時，seteuid を用いている．seteuid は，C/C++の実効ユーザ ID を変更する関数である．この関数を用い，ファイアウォール制御のみ一時的に root 権限を取得する．新たな Opengate では，Node.js を利用してファイアウォールの閉鎖を行う必要があり，その際に root 権限が必要となる．Node.js 上においても従来の Opengate 同様に root 権限の処理を行おうと考えたが，seteuid に相当するものが存在しなかった．setuid の利用は可能であるが，一度 root の権限を破棄すると再取得できなかった．

そこで，Child Process モジュールの fork 関数を利用した．fork は引数にプログラムを与えると，そのプログラムを Node.js のプロセスとして起動する．フォークしてプロセスを起動すると，起動を行った親プロセスの実効ユーザ ID を引き継ぐことができる．

フォークする子プロセスに root 権限を必要とする処理を持つプログラムを指定する．フォーク後，親プロセスは setuid で root 権限を破棄し別の実効ユーザ ID にすることで，メインプログラムを常に root で動かすことを避ける．

権限を必要とする最小の処理を子プロセスに持たせ，親プロセスからの呼び出しによって行う．また，親プロセスと子プロセス間でメッセージのやり取りを行うことができる．これによりメッセージデータにファイアウォールのルール番号を与え，子プロセスでファイアウォールの閉鎖を行った．

3.6 監視方式の切り替え

先にも述べたが，今回の実装は，監視プロセス部分の変更としてシステムを構築しているため，従来の Opengate からの移行が容易である．また，安定した移行が行えるように新たな Opengate は Node.js で監視を行う方式と，従来の方式で行う方式を設定ファイルにより切り替えできるよ

表 1　サーバ PC の仕様
Table 1　Specifications of server PC

OS	FreeBSD 9.0-RELEASE
CPU	Intel® Celeron® CPU 3.06GHz
メモリ	256MB

表 2　クライアント PC の仕様
Table 2　Specifications of client PC

OS	Windows 7 Professional 32 ビット
CPU	Intel® Core™ i7 CPU 870 2.93GHz
メモリ	4GB

う構築した．Node.js の有効・無効を決める EnableNodejs タグを設定ファイルに加え，EnableNodejs の値によって Node.js 方式と監視プロセス方式の切り替え可能とした．

3.7　動作検証

検証を行う環境の構成として，Opengate が動作するサーバ PC 1 台とネットワークを利用するクライアント PC 1 台の 1 対 1 で検証を行った．サーバ PC は，NIC を 2 枚以上持ち，OS は FreeBSD をインストールしたものである．サーバ PC とクライアント PC の仕様をそれぞれ**表 1**, **表 2** に示す．この環境において，WebSocket に関連して以下の項目の動作を確認したが，想定通りの動作が確認できた．

- 認証後のインターネットの利用
- 監視ページ上でのユーザ ID，開始時刻の表示
- 利用終了時のファイアウォール閉鎖
- 利用終了記録のデータベース，ログへの保存

動作確認を行った主なブラウザは，Internet Explorer 9,10，Firefox 19，Google Chrome 25，Safari 5，Opera 12 などであるが，これらのブラウザで正常に動作した．10 時間以上の WebSocket コネクションの維持も問題なく動作した．

4.　まとめと今後の課題

利用の監視に HTTP 通信を使用する従来の Opengate には，コネクションが勝手に切断される場合がある．この問題に対応するため，Node.js を用いたネットワーク利用終了監視システムの開発を行い，WebSocket コネクションの Opengate への組み込みを行った．従来の Opengate 環境に Node.js を加え，Node.js 用のモジュールである Socket.IO を使用し WebSocket コネクションの実装を行った．この実装は，Opengate の監視プロセス部分の入れ替えであり，移行が容易である．また，Node.js を用いた方式へ移行したが安定して運用できなかった場合でも，設定により従来の監視プロセスを使った方式へ戻すことができる．

動作検証により Opengate の認証後の WebSocket コネクションの接続からコネクション切断後のファイアウォール閉鎖，利用終了ログの出力への一連の動作を行えていることを確認した．WebSocket コネクションの検証については，5 つの主要な Web ブラウザで行った．結果，どの Web ブラウザもコネクションが勝手に切れることはなく長時間ネットワークの利用ができることを確認できた．

今後の課題としては，この Node.js による WebSocket を利用した Opengate の全学規模での動作検証などが挙げられる．また，Node.js といった外部の実行モジュールに依存しない WebSocket 通信を行う実装の検討も課題である．

謝辞　WebSocket を用いた Opengate の開発に関して，佐賀大学工学系研究科の中島祥氏，大久保宏倫氏に深く感謝致します．

参考文献

[1] 利用者認証と利用記録機能を実現するゲートウェイシステム Opengate の開発: 渡辺健次, 江藤博文, 只木進一, 渡辺義明, 信学技報 IN99-95, TM99-61, OFS99-48, pp.43-48 (2000)
[2] IPv4/IPv6 デュアルスタックネットワークに対応したネットワーク利用者認証システムの開発: 大谷誠, 江口勝彦, 渡辺健次, 情報処 理学会論文誌, Vol.47, No.4, pp.1146-1156 (2006)
[3] シングルサインオンに対応したネットワーク利用者認証システムの開発: 大谷誠, 江藤博文, 渡辺健次, 只木進一, 渡辺義明, 情報処理学会論文誌, Vol.51, No.3, pp.1031-1039 (2010)
[4] HTTP Keep-Alive による利用終了検知機能を実装した新しい Opengate の開発: 大谷誠, 江藤博文, 渡辺健次, 只木進一, 渡辺義明, 情報処理学会研究報告, 2007-DSM-44, pp.65-70 (2007)
[5] The WebSocket Protocol, RFC6455
[6] The WebSocket API, 入手先 ⟨http://www.w3.org/TR/websockets/⟩
[7] Node.js, 入手先 ⟨http://nodejs.org/⟩
[8] Node.js 日本ユーザグループ, 入手先 ⟨http://nodejs.jp/⟩
[9] Socket.IO, 入手先 ⟨http://socket.io/⟩