低遅延 Web 実況チャットを実現する P2P 網を用いたコンテンツ配信手法

高 橋 明 生[†1]

本稿では，クライアント側の P2P (Peer-to-peer) 網によって遅延時間の短縮と Web サーバの負荷低減を両立させた Web 実況チャット向けコンテンツ配信手法を提案する．近年，テレビ番組の視聴者がインターネットを介してリアルタイムに感想を共有する楽しみ方（実況チャット）が知られるようになった．特に Web サーバ上での実況チャットは古くから行われており，現在でも多くの参加者がいる．しかし，HTTP (Hyper-text Transfer Protocol) はプッシュ型の通信に適しておらず，サーバ負荷を抑えながら低遅延でデータを配信することが難しい．そこで提案手法では，P2P 網を用いてクライアント間で最新データを共有し，さらに P2P 網全体が効率的に動作するようにした．検証の結果，16 クライアントが 30 秒周期でデータを取得するとき遅延時間が 2 秒程度であるなど，効果を確認できた．

Contents Delivery Method with P2P Network to Reduce Latency on Web Live Chatting

Akio Takahashi[†1]

We proposed a contents delivery method to reduce latency for web live chatting. Live chatting is the communication activity amoung many users watching same contents such as TV program. Some live chatting communities exist on Web servers but there is a issue, trade-off between latency and server load. In our method, clients share new posted data by using a peer-to-peer network and the Controllers (super peers) coordinate client's retrieving timing. We confirmed 2 second latency in the condition that 16 clients retrieve posted data once every 30 seconds.

†1 株式会社豆蔵
Mamezou Co., Ltd.

1. 序 論

テレビ番組などの視聴において，視聴者同士がリアルタイムに感想を共有したいというニーズがある．実際，従来から飲食店でのテレビモニタの設置や街頭の巨大スクリーンへの投影などが行われており，さらに近年ではインターネットを利用したコミュニケーション手段が提供されるようになった．特に，テキスト主体の双方向コミュニケーション（実況チャット）は一般家庭のインターネット回線が低速だった 2001 年頃にはすでに存在し，現在まで活発に利用されている．実況チャットでは，感想を短い「コメント」に表し，参加者間で送信し合う．

広く受け入れられている実況チャットの形態として，本研究の対象である Web サーバ上で行う実況チャット（**Web 実況チャット**）がある[2]．Web サーバで運用することで，1) 低料金または無料のレンタルサーバを利用できる，2) Web ブラウザ以外の特別なプログラムが不要で利用しやすい，という利点がある．

Web 実況チャットの典型的な運用システムは電子掲示板システム (BBS: Bulletin Board System) である．BBS では，参加者がサーバにコメントを投稿でき，投稿されたコメントは直ちに他の参加者から読めるようになる．

1.1 Web サーバ上の実況チャットにおける問題

Web サーバでは通信に HTTP (Hyper-text Transfer Protocol) を用いるが，特に多人数が参加する状況において，HTTP は**遅延時間**と**サーバ負荷**という点で実況チャットに適していないという問題がある．ここで，遅延時間とは，コメントが Web サーバに投稿されてからユーザに届くまでの時間とする．

Web 実況チャットを行う場合，クライアントは短い間隔でサーバをポーリングし，最新コメントを取得する．HTTP はプル型であるため，クライアントはポーリングしなければ最新コメントの有無を知ることができない．また，コメントが投稿されてからユーザに届くまでの時間は短いほうが望ましいため，ポーリング間隔を短くして遅延時間を短縮する．

しかし，高頻度のポーリングによるサーバ負荷の増大は無視できない．ポーリングでは最新コメントがない場合でもサーバに問い合わせるため，間隔を短くすると無駄な通信が頻発してしまう．また，ピーク負荷と平常負荷の差が大きいため，サーバのマシンや回線の高性能化による対策は資源効率やコストの点で最適ではない．

1.2 既 存 技 術

サーバのコンテンツを多数のユーザに配信する技術がすでにあるが，コストやサーバの変

更が必要である点が Web 実況チャットには適していない．特に，全ユーザに影響するサーバ側の変更は避ける必要がある．したがって，既存 Web サーバに適用可能な低遅延かつ低負荷なコンテンツ配信手法が求められている．

1.2.1 サーバプッシュ技術

遅延時間を短縮する技術として，Web サーバでプッシュ型通信を実現する技術（HTTP Server Push）がある[3]．

しかし，サーバプッシュ技術はサーバ側の対応が必要であり，一般的な Web サーバでは利用できない．また，クライアントとの接続を常に維持するため，サーバ負荷を増加させるという問題がある．

1.2.2 代理サーバの導入

サーバ負荷を低減する方式として，クライアントの代わりに Web サーバへアクセスするコンピュータ（代理サーバ）を設ける方式が考えられる．

しかし，代理サーバを利用する方式では新たにコンピュータや通信回線を設置する必要があり，コストや手間が掛かる．

1.2.3 P2P コンテンツ配信ネットワーク

一般に，コンテンツ配信を目的とするネットワークを CDN (Contents Delivery Network) と呼ぶが，CDN と P2P (Peer-to-peer) 技術を組み合わせて配信サーバの負荷低減を図った技術が複数存在する[5]．これらの技術では，配信対象のコンテンツを複数の中継サーバに複製しておくことで，多数のクライアントからの要求に応える．

しかし，CDN の技術は動画などの大きなデータに対して効率的であるとされており，実況チャットのデータ（ほとんどが数行のテキストデータ）を想定していない．また，データがリアルタイムに追加され，かつ僅かな時間で需要が低下するといった特徴にも，既存の CDN 技術では対応しにくいと考えられる．

1.2.4 Web コンテンツキャッシング P2P ネットワーク

Web サーバのコンテンツをキャッシュする P2P 網を構築し，クライアントに対してプロクシのように振る舞うことで Web サーバの負荷を分散させる技術も存在する[1]．

しかしながら，実況チャットではコンテンツが秒単位で更新されることに加え，キャッシュを長期間保有しておくことも必要ない．したがって，実況チャットのようなコンテンツに対しては，キャッシュデータのヒット率や検索性を追求するよりも遅延時間を短縮することに重点を置いた方式が必要である．

2. Web 実況チャット向けコンテンツ配信方式の提案

前述の問題を解決するため，我々は，サーバの最新情報を P2P 網に参加したクライアント間で共有することによって遅延時間とサーバ負荷を改善する方式を提案する．Web 実況チャットではサーバからクライアントへの通信がほぼ同じデータの繰り返しであることに着目し，繰り返し部分をクライアントが高速に共有できれば低遅延と低負荷を両立できると考えた．

また通常，P2P 技術によるデータ共有ではデータの検索や一貫性の確保が課題になるが，Web 実況チャットでは 1) データがサーバで一元管理されている，2) 各クライアントは同一のコンテンツを受け取る，3) 数分経過したコンテンツは再利用されない，という性質があり，これらの問題を回避できると考えた．

提案方式では，遅延時間が 2 秒のときのサーバ負荷が現状方式と比べて低下することを目標とした．理由は，テレビ番組の中継におけるアナログ放送とデジタル放送の時間差が約 2 秒であり，実況チャットにおいて目立った問題になっていないためである．

2.1 提案方式の課題

提案方式によって所望の効果を得るためには，以下の課題を解決する必要がある．

クライアント数に対する規模追従　実況チャットでは，コンテンツによって参加者の数が数人から数百人まで変化する．したがって，参加数が少ない状況でも効果が得られ，さらに参加者の増加に追従して性能を改善できる必要がある．

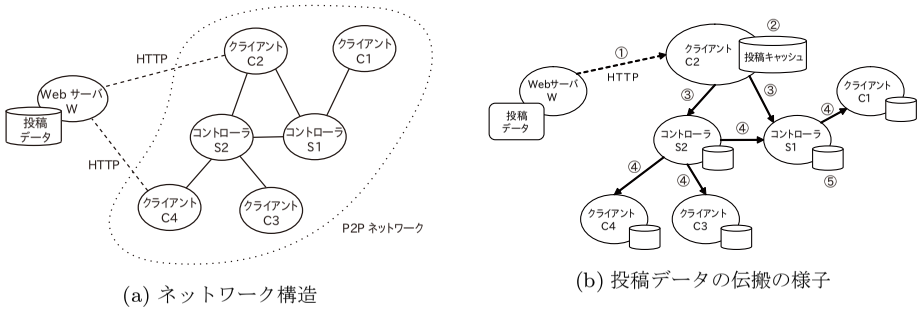
クライアントの動的増減への対応　クライアントは P2P 網に動的に参加したり離脱したりする．特に，コンテンツの切れ目（テレビ番組時間帯の境目など）は一斉に変化することがあり，この変化に耐えられる必要がある．

サーバへの問い合わせ回数の低減　クライアントのサーバアクセス動作を制御し，サーバに与える負荷を低減する必要がある．本研究では，P2P 網全体でみたときのクライアントからサーバへの問い合わせ回数を指標とし，これを現状と比べて低減する．

3. 解 決 手 法

我々は前述の課題について，P2P 網に特別なピア（コントローラ）を配置し，クライアントの動作を制御する手法によって解決した．以降，本手法について説明する．

なお，クライアントという言葉には，本手法における「クライアントピア」と「HTTP クライアント」の両方の意味がある．しかし，後述するように，本手法のクライアントピア



(a) ネットワーク構造　　(b) 投稿データの伝搬の様子

図 1　提案手法のシステムにおけるネットワーク構造と振る舞い
Fig. 1　Network Structure and Behavior of Proposed Method

は Web サーバの HTTP クライアントとしても振る舞うため，本手法においては両者を等しいものと扱って差し支えない．

3.1　システムの構成

本手法を適用したコンピュータシステムは，サーバ，クライアント，コントローラから成り立つ（**図 1**-a 参照）．クライアントとコントローラをまとめてピアと呼ぶことにする．

Web サーバ W は，BBS などの Web 実況チャットシステムを稼働しているサーバである．実況チャットの参加者は Web ブラウザでコメントを投稿できる．参加者が書き込んだコメントは，サーバから次の情報を含む**投稿データ**として取得することができる．

- コメント内容（投稿者名や本文など）
- コメントの書き込み時刻

クライアント $C_{1,2,3,4}$ とコントローラ $S_{1,2}$ は，ユーザのコンピュータで動作するプログラムである．ここで，ユーザとは本手法を実装したプログラムを利用している参加者とする．

クライアントの役割は，P2P 網へサーバ W 上の投稿データを供給することと，ユーザに対して受信したコメントを表示することである．コントローラの役割は，クライアントからの接続を受け付け，接続中のクライアントを制御することである（スーパーピア）．クライアントとコントローラは，同一のコンピュータ上で動作していてもよい．

3.2　システムの動作

3.2.1　P2P 網上での投稿データの伝搬

図 1-b を参照しながら，P2P 網に投稿データが伝搬する過程を説明する．

P2P 網へ投稿データを供給するのはクライアントである．クライアントはサーバに問い合わせを行い，最新の投稿データを取得する (1)．つぎに，ピアごとに備えている投稿キャッシュを用いながら投稿データの転送判定を行う (2)．転送判定の詳細は 3.2.2 で後述する．

投稿データの転送が必要な場合，接続中のコントローラへ投稿データを送信する (3)．コントローラは投稿データを受信すると，転送判定を行い，必要な場合は他のピア（クライアントまたはコントローラ）に転送する (4)．以降，中継先のピアも同様に処理し，P2P 網全体に投稿データが伝搬する．

その後，投稿データが十分にネットワークに行き渡ると，ピアが同じ投稿データを複数回受信するようになる．その場合，重複した受信データは転送判定によって再転送が抑止される (5)．なお，転送はコントローラに限らず，クライアントもコントローラから受信した投稿データを他のコントローラへ転送する．

3.2.2　投稿キャッシュを用いた転送判定

各ピアは，直近に処理した投稿データをエントリとするキャッシュ（投稿キャッシュ）を備えている．投稿キャッシュのエントリ数に制限はないが，各エントリは登録から一定時間経過すると消去される．

各ピアでは，サーバまたはピアから受信した投稿データについて次の条件と一致判定を行い，転送の要否を決定する．

(1)　投稿データの投稿時刻が投稿キャッシュの寿命より古い．
(2)　投稿データが投稿キャッシュに存在する．

第一の条件は，投稿キャッシュのエントリに寿命があるため，それを超える投稿データを一律に破棄するための条件である．これにより，無用な情報が P2P 網上を流れ続けるのを防止すると共に，投稿キャッシュのサイズを小さくしている．

第二の条件は，直近で受信した投稿データの再転送を防止する．投稿キャッシュにヒットした投稿データは転送済みであり，ピアはその投稿データを破棄する．

3.2.3　コントローラによる取得タイミングの調整

以上に述べた転送動作によって，各ピアはサーバに問い合わせを行うことなく，最新の投稿データを P2P 網から受信できる．

しかしながら，各クライアントがサーバから投稿データを取得するタイミングが悪いと，効率的に P2P 網を活用できない．たとえば，全クライアントが同時に取得している状況では，P2P 網に送信される投稿データはどのクライアントも不要であり，通信は無駄になる．ピア間の通信時間を無視すれば，P2P 網全体でいずれかのクライアントが一定間隔でサーバへ投稿データを問い合わせている状態が最も効率がよい．

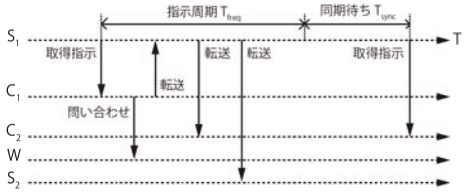


図 2 コントローラによるクライアントの取得タイミング調整
Fig. 2 Timing Synchronization during Clients by Controller

そこで本手法では，コントローラがクライアントに投稿データの取得タイミングを指示することで，P2P 網に効率よく投稿データが供給されるようにした．コントローラからクライアントへの指示動作は，取得指示ステップ，待機ステップ，同期ステップで構成され，この 3 ステップを繰り返す．

以降に，**図 2** を参照しながら，図 1 のコントローラ S_1 を例に指示動作を説明する．

3.2.4 取得指示ステップ

取得指示ステップは，接続中のクライアント (C_1, C_2) の中から，最も長い間投稿データを送信していないもの（C_1）を選び，取得指示を送信する手順である．

取得指示を受信したクライアント C_1 は直ちにサーバ W へ問い合わせを行い，最新の投稿データを取得し，他のピア (S_1) へ転送する．

3.2.5 待機ステップ

待機ステップは，最後の取得指示ステップから指示周期 T_{freq} だけ待機する手順である．指示周期 T_{freq} は式 1 で計算する．ただし，Nc は接続中のクライアントの数，T_{min} は最短指示周期，T_{max} は最大指示周期である．

$$T_{freq} = T_{min} + \frac{T_{max} - T_{min}}{Nc} \tag{1}$$

この式に従うと，接続クライアント数が 1 のときの指示周期が T_{max} となり，クライアントが増加するにつれ T_{min} に近づく．一方，各クライアントにおける取得間隔は長くなる．

3.2.6 同期ステップ

同期ステップでは，さらに同期待ち時間 T_{sync} だけ待機する．T_{sync} は，待機ステップ中に他のコントローラから転送されてくる投稿データの受信タイミングを元に計算する．この計算について**図 3** を参照しながら説明する．

図 3 は，あるコントローラが，前回の取得指示時刻 T_0 から現在時刻 T_{now} までに，自コントローラが別のコントローラ S_x および S_y から投稿データを受信していた様子である．T_{first} は前回取得指示のあと他のコントローラから最初に投稿データを受信した時刻，T_{last} は最後に受信した時刻である．I_0 および I_1 は次のように計算する．

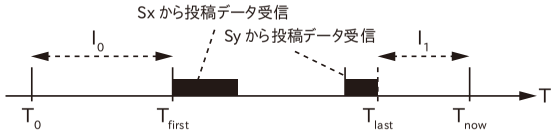


図 3 コントローラ間のタイミング調整
Fig. 3 Timing Adjustment between Controllers

$$I_0 = T_{first} - T_0 \tag{2}$$

$$I_1 = T_{now} - T_{last} \tag{3}$$

I_0 と I_1 が等しいときは複数のコントローラの取得指示タイミングは望ましい状態であるが，$I_0 > I_1$ であるときは自コントローラの取得指示タイミングを遅めたほうが良いことを示している．そこで，I_0 と I_1 から修正量 I_{mod} を式 4 のように計算し，式 5 のように T_{sync} を決定する．ただし，$rand(x, y)$ は区間 $[x, y)$ の一様分布乱数，k は $0 < k \leq 1$ の修正量係数である．

$$I_{mod} = (I_0 - I_1)/2 \tag{4}$$

$$T_{sync} = \begin{cases} rand(0,1) & \text{if } I_0 \simeq 0. \\ kI_{mod} & \text{if } I_{mod} > 0. \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

I_{mod} は，正であれば自コントローラの取得指示を遅くすべきであり，負であれば早めるべきであることを示す．本手法では，遅くするべき場合にのみ kI_{mod} だけ修正するようにした．係数 k を乗じることで徐々に同期するようにしている．

また，I_0 がほぼ 0 であるときは他のクライアントと同時に取得していると考えられるため，1 秒未満のランダムな時間だけ待機し，タイミングをずらすようにした．

4. 動作検証

提案手法を実装したプログラム（ピアエミュレータ）と実況チャット Web サーバのエミュレータ（サーバエミュレータ）を開発し，シミュレータによる仮想環境内で動作を検証した．

4.1 検証環境

ピアエミュレータは，コントローラとクライアントを任意の数だけ実行でき，かつ任意のピア間を接続できる．提案手法におけるコントローラの動作パラメタは $T_{min} = 1, T_{max} = 12, k = 0.5$ とした．サーバエミュレータは，1 台の実況チャット Web サーバを模擬する．このサーバには，1 秒間に 1 件の投稿が自動的に生成される．

これらのプログラムをシミュレータに接続し，Windows 上で動作させモニタリングした．ただし，仮想環境内のノード（ピアとサーバ）の処理性能は無限であり，すべての処理が直ちに完了する．また，ノード間の通信時間は固定であり，クライアントがサーバから投稿データを受信するには，サイズや件数に依らず 100ms 掛かる．ピア間の投稿データの転送には，サイズによらず一件あたり 50ms 掛かる．

これらより，仮想環境内では通信のみが時間経過の要因となる．これは現実にはあり得ないが，家庭用コンピュータであっても通信処理にかかる時間は ms 単位であり，検証においては通信時間に合算して考慮すれば十分と考えた．

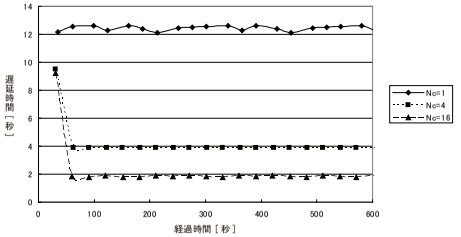
4.2 クライアント数と遅延時間の関係

提案手法による遅延時間の短縮効果を調べるため，仮想環境に配置するピアを変化させてクライアントにおける遅延時間の変化を測定した．遅延時間は，30 秒の区間ごとに，区間中に受信した最も古い投稿データの投稿時刻と受信時刻との差とした．したがって，遅延時間の最小は 0（投稿直後に受信），最大は前回取得時刻からの経過時間（前回取得直後に投稿）である．遅延時間がより小さい方が投稿データを直ちに受信していることになる．

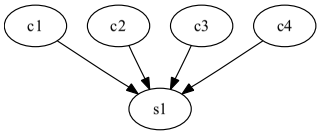
図 4-a は，コントローラ数 $N_s = 1$ と固定し，クライアント数 N_c を 1, 4, 16 としたそれぞれの場合における 10 分間の遅延時間の変化である．図 4-b は $N_c = 4$ の場合の P2P 網の構造である．ただし，c1, c2, c3, c4 はクライアントであり，s1 はコントローラである．(以降の図でも，X を番号とし，cX はクライアントを，sX はコントローラを表すものとする．なお，ピア間リンクの矢印は接続方向であり，通信は双方向に行える.)

図 4-a から，クライアント数が多い方が遅延時間が短いことが分かる．$N_c = 1$ の場合には遅延時間がゆらいでいるが，これはサーバの書き込みデータの生成タイミング（秒間一件）とのずれが影響しているためである．

図 5-a は $N_c = 16$ と固定し，N_s を変化させた場合での 10 分間の遅延時間の変化である．ただし，クライアントは各コントローラに均等に割り振った．また，コントローラ間は各々と直接接続し，クライアントはいずれか一つのコントローラに接続するようにした．図 5-b に $N_s = 4$ の場合の P2P 網の構造を示す．

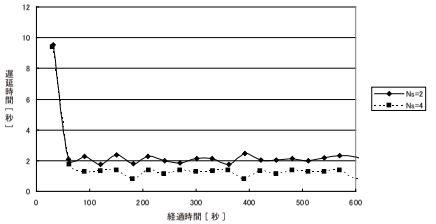


(a) 投稿データの伝搬の様子

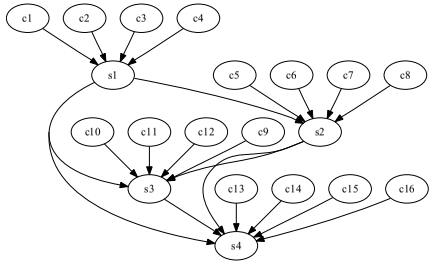


(b) $N_c = 4$ でのネットワーク構造

図 4 クライアント数と遅延時間の関係
Fig. 4 Relation between Number of Clients and Latency



(a) 遅延時間の変化



(b) $N_s = 4$ でのネットワーク構造

図 5 コントローラ数と遅延時間の関係
Fig. 5 Relation between Number of Controllers and Latency

図 5-a より，ピアが複数のコントローラに分散した場合でも，遅延時間が短縮されていることが分かる．$N_s = 4$ の場合より $N_s = 2$ の場合の方が遅延時間が長いが，これは接続中のクライアントが多いと取得間隔が長くなるためである（式 5 参照）．

4.3 クライアントの動的な増減による遅延時間の変化

クライアントの動的な増減による遅延時間への影響を調べるため，動的にクライアントを接続・切断し，C_1 の遅延時間を測定した．結果を**図 6** に示す．

まず，$Ns = 1$ の場合は，初期状態で S_1 に C_1 が接続しており，5 分経過時点でに C_2 が接続，10 分経過時点で $C_{3,4}$ が接続し，さらに 15 分経過時点で $C_{5,6,7,8}$ が接続する．その後，20 分経過時点で $C_{5,6,7,8}$ が切断，25 分経過時点で $C_{3,4}$ が切断し，最後に 30 分経過時

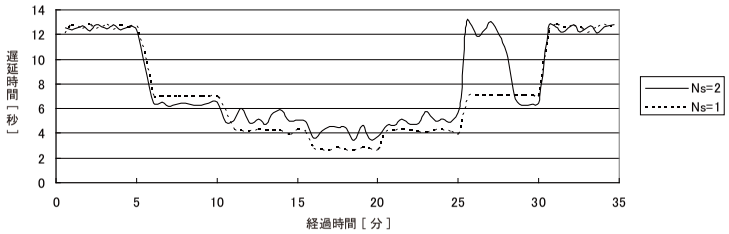
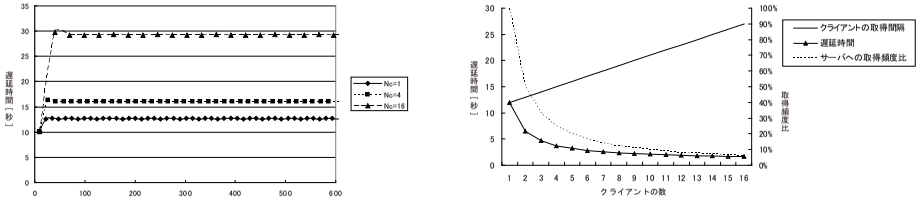


図 6 コントローラが 1 個と 2 個の場合における遅延時間
Fig. 6 Comparison of Letency with Single Controller and Two Controllers



(a) クライアントからの問い合わせ間隔　(b) クライアント数と投稿データ取得頻度比の関係

図 7 提案手法による問い合わせ回数の低減
Fig. 7 Reduction of Requests by Proposed Method

点で C_2 が切断する．35 分経過時点までシミュレーションした．

$Ns = 2$ の場合は，各クライアントが接続・切断するタイミングは $Ns = 1$ と同様であるが，S_1 に加えて S_2 を設置し初期状態で両ピアを接続した．さらに，$C_{2,5,6,7,8}$ の接続先は S_2 とした．すなわち，クライアントの数は $Ns = 1$ と同じだが，接続先を交互にした．

まず $Ns = 1$ の場合，クライアントの増減に対応できており遅延時間が安定していることが分かる．$Ns = 2$ の場合でも，ほぼ $Ns = 1$ の場合と同様の遅延時間を得られているが，25 秒経過時点で大きく延びている．これは $C_{3,4}$ の切断でコントローラの取得指示タイミングが揃い，コントローラ $S_{1,2}$ 間で上手く協調できていないことを表している．しかし，同期処理により，約 3 分程度で再び遅延時間が減少し，効率的に動作している．

4.4 サーバ負荷の低減

サーバ負荷について調べるため，1 個のコントローラに接続するクライアント数 N_c の変化によるサーバへの問い合わせ間隔を検証した．

図 7-a には測定した問い合わせの間隔を，図 7-b には計算で求めた遅延時間と取得間隔，および提案手法を用いない場合との投稿データ取得頻度の比率を示す． 図 7-a より，クライアント数が増加すると取得間隔が長くなる様子が分かる．図 7-b より，16 ピアの接続で取得間隔が 27 秒になり，本提案手法を用いない場合との頻度比が 6.3%になる．

5. 評　　価

検証結果より，サーバ負荷を低減しつつ遅延時間を 2 秒以下にするという目標を達成できた．今後，数百ピアでの検証が必要であるが，その場合でも今回検証した構造を複数接続すれば規模追従すると期待できる．

以上より，提案方式は，既存の Web サーバに適用可能であり，Web 実況チャットを従来より低負荷かつ低遅延にするといえる．

6. 結　　論

本稿では，Web 実況チャットを対象に，ユーザ側で P2P 網を構築して最新のコメントを共有する方式を提案した．また，P2P 網を制御するピア（コントローラ）によって P2P 網全体での遅延時間短縮とサーバ負荷低減を図る手法を提案し，シミュレーションによる検証で効果を確認した．

今後は，P2P 網の最適な構築手法を研究予定である．P2P 網のトポロジーが最適でない場合，遅延時間が徒に悪化する懸念があり，P2P 網を動的に再構築すれば効率のよい配信が可能になると考えられる．また，実際の Web 実況チャットシステムは，ユーザの嗜好に合わせてサブカテゴリを持つ場合が多い．サブカテゴリごとに本提案手法を適用可能か，もしくは P2P 網でのルーティングを検討する必要があると考えている．

参　考　文　献

1) Freedman, M., Freudenthal, E., Mazieres, D. and Eres, D.M.: Democratizing content publication with Coral, *In NSDI*, pp.239–252 (2004).
2) PACKET MONSTER INC.: 2 ちゃんねる，http://www.2ch.net/ (2009).
3) Russell, A.: Comet: Low Latency Data for the Browser, http://alex.dojotoolkit.org/?p=545 (2006).
4) ジー・モード：おしゃべりテレビ，http://www.oshaberitv.com/ (2009).
5) 市川　類：コンテンツ配信 (CDN) 技術と P2P 技術を巡る動向 (2009).