東京大学
情報理工学系研究科 電子情報学専攻
修士論文

クライアントサイドにおける 異なるメディアストリームの再生タイミング同期技術

A Client Side Technology of Timing and Synchronization between Different Media Streams

48-126431

中村 哲也

Tetsuya Nakamura

指導教員　江崎 浩 教授

2014 **年** 2 **月**

概要

近年のインターネットの普及によって数多くの新しいインターネットメディアが次々と出現している．具体的には Twitter や facebook などの SNS，YouTube や Ustream などの動画配信サービスが挙げられる．これらのインターネット上で展開されるさまざまなメディアを連携したサービスも出現している．メディアストリームの連携と同期はこのような複数のインターネットメディアを連携させた新しいサービスの実現に必須のものである．インターネットメディアの連携・同期は，サーバ・プロキシ・クライアントの 3 つの内どこかで行う必要があり，どこで行うかによってサービスや技術，そして実現のためのコストが変わる．サーバで同期を行うのは対象とするメディアストリームが制限されることや，サーバリソースの消費，さらに，サービスの拡張変更の自由度の減少という問題をもつ．また，プロキシでの同期はプロキシという物理リソースを消費してしまうことや普及のために必要な機器の設置コストが大きくなってしまうという問題がある．一方で，近年のパーソナルコンピュータの性能向上によりクライアントの処理能力はますます向上しており，同期処理をクライアントサイドで行うことが望ましくなっている．また，サーバサイドでの連携・同期と異なり，より多くのメディアストリームを，より自由に連携させることを可能とすることができる．そこで本研究では，メディアストリームの中でも時間に敏感な，動画と SNS の 1 つである Twitter に注目し，この 2 つのインターネットメディアをクライアントサイドで連携，同期する手法を提案，実装，その有効性を示した．

目次

第 1 章　序論 1
1.1　背景 1
1.2　本論文の目的 4
1.3　本論文の構成 4

第 2 章　動画配信技術 5
2.1　サービスの形態 5
2.2　配信技術 6

第 3 章　動画と SNS の連携 11
3.1　Hybridcast 11
3.2　連携において発生する問題 11
3.3　問題への対策 13

第 4 章　提案手法 15
4.1　想定環境 15
4.2　機能要件 16
4.3　実装 17

第 5 章　実験・評価・考察 21
5.1　実験 21
5.2　評価 27
5.3　考察 29

第 6 章　結論 33
6.1　まとめ 33
6.2　今後の課題 33

参考文献 35

図目次

1.1 全世界のインターネットコンシューマ IP トラフィック 2
1.2 異なるメディアストリームの連携:サーバサイド 3
1.3 異なるメディアストリームの連携:プロキシ 3
1.4 異なるメディアストリームの連携:クライアントサイド 3

2.1 MPEG-DASH の動作 9

3.1 動画遅延に起因するツイートの遅延の例 12
3.2 クライアントサイドにおけるメディアストリームの同期の例 13

4.1 想定環境におけるグループの定義 16
4.2 動画の再生位置と現在時刻の関係 (A) 18
4.3 動画の再生位置と現在時刻の関係 (B) 19

5.1 ネタバレ防止実験の環境 23
5.2 提案手法適用前の無線環境実験測定結果 (A) 24
5.3 提案手法適用後の無線環境実験測定結果 (A) 25
5.4 提案手法適用前の帯域制限された有線環境実験測定結果 (A) 25
5.5 提案手法適用後の帯域制限された有線環境実験測定結果 (A) 25
5.6 提案手法適用前の無線環境実験測定結果 (B) 25
5.7 提案手法適用後の無線環境実験測定結果 (B) 26
5.8 提案手法適用前の帯域制限された有線環境実験測定結果 (B) 26
5.9 提案手法適用後の帯域制限された有線環境実験測定結果 (B) 26
5.10 天空の城ラピュタの「「バルス」」に誘発される Twitter のツイート 31
5.11 高速再生機能による遅延削減実験の測定結果:バッファ 32
5.12 高速再生機能による遅延削減実験の測定結果:スループット 32

表目次

5.1 実験に使用したマシンのスペック 22
5.2 ネタバレ防止実験の測定結果 23
5.3 動画遅延削減実験の測定結果 24
5.4 動画 1 分後のシーンに対するツイートの遅れ (秒) 27
5.5 高速再生機能による遅延削減実験の測定結果 31

第 1 章

序論

本章では本研究の背景，目的と本論文の構成を述べる．

1.1 背景

近年，インターネットが普及しその技術が飛躍的に向上していることによってブロードバンドサービスを利用できるユーザが急激に増えてきた．それにともない YouTube[1]，ニコニコ動画，ustream，skype といった動画配信サービスも増加している．Cisco の発表によると 2013 年に全世界のコンシューマインターネットトラフィックに占めるインターネットビデオトラフィックの割合は 38.3% であり，IP VoD やビデオストリーミングを用いたゲームなどのトラフィックも含めると 60% である．さらに 2017 年には 52% になる見込みであり，P2P など全ての形式も含めると約 73% になると予測されている [2](図 1.1).

動画の他にユーザ数が増加しているものとしてソーシャルネットワーキングサービス (SNS) が挙げられる．米 Twitter 社が 2013 年 10 月に提出した IPO 申請書類である Form S-1[3] によると，2013 年 4 月から 6 月の月間平均アクティブユーザ数は 2 億 1830 万人となっている．2011 年の 8500 万人，2012 年の 1 億 5000 万人と比較すると，アクティブユーザが着実に増加していることがわかる．また，他のサービスとして facebook や google+ などが挙げられる．facebook は 2013 年 3 月時点で月間アクティブユーザ数 11 億人，google+ は 2013 年 10 月時点で 3 億人と発表されており，非常に多くの人間が SNS を利用していることがわかる．SNS の他にも 2 ちゃんねるのような電子掲示板もオンラインでのコミュニケーションツールとして利用されており，インターネットを通じたコミュニケーションは日常でありふれたものとなっている．

上記で挙げたような動画配信やオンラインコミュニケーションのサービスが広く普及してきたことによって，複数のメディアストリームを連携，同期して楽しむサービスも出現してきている．メディアストリームの連携，同期は図 1.2 のようなサーバ同期型，図 1.3 のようなプロキシ同期型，図 1.4 のようなクライアント同期型の 3 種類が考えられる．サーバで同期を行うのは同期対象とするメディアストリームが制限されることや，サーバリソースを消費してしまうという問題がある．また，プロキシでの同期はプロキシという物理リソースを消費してしま

うことや，デプロイコストがかかるという問題がある．一方，近年はパーソナルコンピュータの性能が向上しているためクライアントでの処理能力が高くなっており，クライアントサイドで同期処理を行うことが望まれる．また，サーバサイドでの同期と異なり，同期対象とするメディアストリームが幅広いというメリットもある．

メディアストリームの同期という点では，時間に敏感なメディアである動画のリアルタイム配信や Twitter などのマイクロブログが注目される．実際，ユーザがインターネット上で動画を見ながら同時にインターネットを介してコミュニケーションをとるという機会が増えてきている．図 1.2 のようなニコニコ生放送 [4] や Ustream[5] などの動画配信とコミュニケーションの場が同じサービス上にあるものであれば，コメントとタイムスタンプを結びつけることによって全ユーザに同じ再生時刻にコメントを見せることが可能である．しかし図 1.4 のような動画配信とコミュニケーションの場が別のサービス上にある場合，例えば YouTube のライブ配信を見ながら Twitter で実況をするような場合は，動画データが各ユーザに到着する時間にずれが生じることにより，早く到着したユーザの反応が SNS などに書きこまれ遅く到着したユーザへのネタバレが生じてしまう恐れがある．また，動画データが遅く到着したユーザの反応は早く到着したユーザにとっては動画のシーンに対して遅れる．そのため同時に動画を見ているはずなのにライブ感が得られないという問題がある．

韓国では IPTV によるテレビの動画コンテンツ再配送サービスが普及しており，アメリカでは Netflix などの動画配信サービスが普及している．また日本でも実際にニコニコ生放送で放送されたコンテンツを後に地上波で放送するというケースも現れている．総務省の放送サービスの高度化に関する検討会 [6] では 4K/8K 放送を実現するための伝送路として IP 網も含まれており，日本のテレビでも動画を IP ネットワークを通じて配信する可能性が高い．このようにして IP ネットワークを通じた動画配信が普及することで動画の到着がずれる問題が顕在化すると予想される．

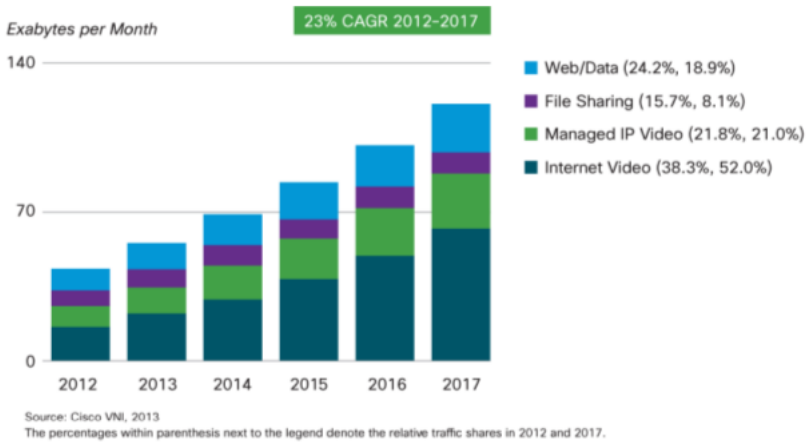

図 1.1. 全世界のインターネットコンシューマ IP トラフィック

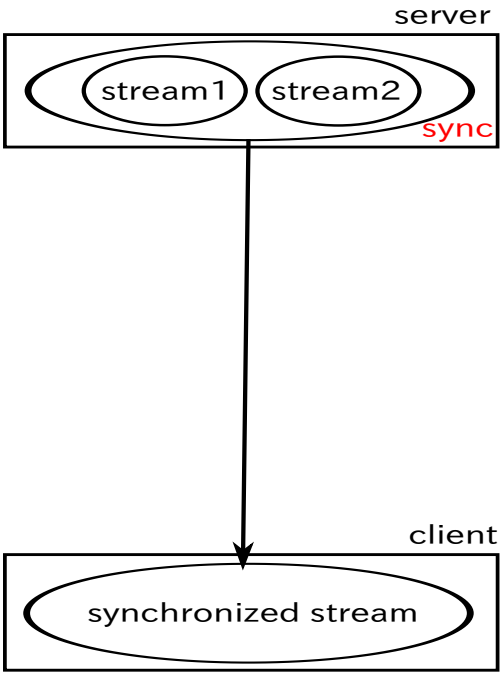

図 1.2. 異なるメディアストリームの連携: サーバサイド

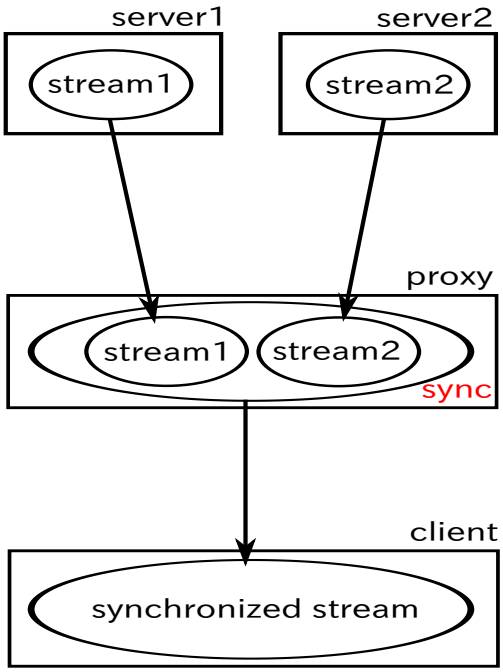

図 1.3. 異なるメディアストリームの連携: プロキシ

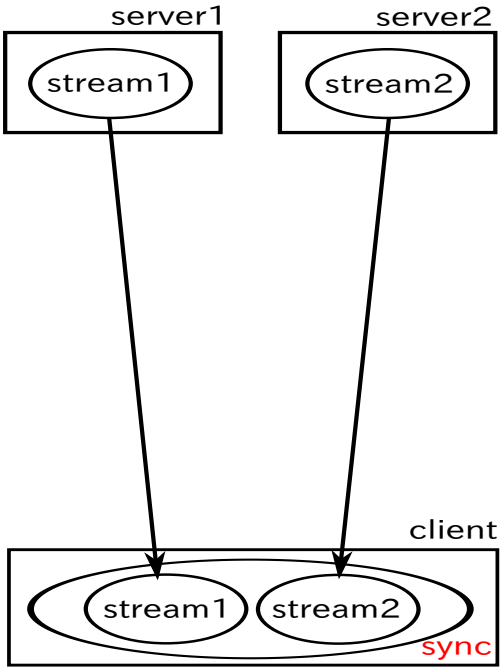

図 1.4. 異なるメディアストリームの連携: クライアントサイド

1.2　本論文の目的

本研究では，図 1.4 のようにクライアントサイドで異なるメディアストリームの再生タイミングを同期することを目的とする．また，メディアは時間に敏感な動画と Twitter を使用する．つまり，クライアントサイドで時刻同期し，それらのずれを小さくすること及びネタバレを防ぐことを目的とする．クライアントサイドで同期するのは，サーバサイドでの同期はメディアストリームを提供するサービスが同じサービス上になければならないという制約があるのに対し，クライアントサイドでの同期ではその制約がないからである．そうすることによって，異なるサービス・ストリームからのものであっても連携したサービスを提供可能とすることを目指す．これらの目的を実現するために必要な機能要件を整理し，それを満たす実装を提示する．その上で実装したプログラムで実験をし，提案手法を適用した場合としていない場合のずれを比較，問題点をどの程度解決できているかを評価する．ずれの評価については実験結果が実際の Twitter のデータに対してどれくらいの遅延になっているかを指標とする．

1.3　本論文の構成

本論文は全 7 章から構成される．本章が第 1 章であり，第 2 章では動画配信技術をいくつか挙げ，本研究で用いる技術を紹介する．第 3 章では本研究の主な目的である動画と SNS コンテンツの連携について説明し，第 4 章で提案手法について述べる．第 5 章において実験について述べ，第 6 章でその結果を評価する．そして第 7 章で実験結果及び評価の考察を述べ，最後に第 7 章で本研究の結論をまとめる．

第 2 章

動画配信技術

ネットワークの帯域や伝送速度が向上することによって動画のような大容量のデータを配信が可能となった．それにともないニコニコ動画や YouTube のような動画配信サービスが普及してきている．本章では，そのような動画配信サービスで用いられている技術を整理する．

2.1 サービスの形態

動画配信技術は使用されるサービスによって異なる．そこで，まずは動画配信サービスにどのような形態があるかを述べる．

2.1.1 ビデオ・オンデマンド

ビデオ・オンデマンド (VOD) とは，ユーザが観たいとき観たい動画を視聴することが可能なサービスである．具体的なサービスとして YouTube やニコニコ動画がある．配信されるコンテンツは公開済みの映画や放送済みのテレビ番組のように，既に用意されたものが使用される．またニコニコ動画や YouTube は商用に制作されたものでない個人が制作した動画もコンテンツとして配信している．いずれにせよ予めエンコードして用意された動画データを使用する．

ビデオ・オンデマンドにおいて，商用に用意されたコンテンツの視聴には課金システムを導入するのが一般的である．商用に用意されたコンテンツとは，例えばドラマや映画，アニメーションなどの我々がテレビや映画館で見ているコンテンツを指す．課金形態には大きく 2 つの形態があり，pay-per-view と定額制とがある．pay-per-view とは，各コンテンツにつき料金の支払いが発生するものである．ユーザは見たい番組があればその都度その番組に設定された料金を支払う．そのため，普段あまり動画を見ないユーザにとっては支払う料金が少なくて済むというメリットがある．一方定額制は，ある期間に設定された料金を支払えばその期間はどのコンテンツも見放題となる形態である．ただし，サービスによっては期間内に視聴できるコンテンツの数に限りがある場合もある．こちらは一定額を支払えば多くのコンテンツを見ることが可能となるため，普段動画をよく見るユーザにとっては pay-per-view よりも 1 本あたり

に支払う金額が少なくて済むというメリットがある．多くのサービスでは両課金形態をサポートしている．

2.1.2　ライブストリーミング

ライブストリーミングとは，カメラで撮影した映像・音声をリアルタイムでエンコードしライブ配信するサービスである．具体的なサービスとして Ustream や YouTubeLive がある．ビデオ・オンデマンドとは異なり，動画データはリアルタイムで生成される．ライブストリーミングのメリットはその即時性にあり，テレビの中継のような体験をユーザに提供することが可能となっている．また，テレビはテレビ局のような権利を持った団体のみが配信可能であるのに比べ，ライブストリーミングはネットワーク環境と録画，配信するインフラさえ整えれば誰でも配信することが可能である．そのため，テレビ番組と同じようにコンテンツを配信する試みも行われている．Internet Initiative Japan (IIJ) は 2009 年の皆既日食，2013 年夏の甲子園などをライブストリーミングで配信しており，甲子園の決勝戦では同時視聴者数 11.8 万人，80Gbps を記録した [7].

2.2　配信技術

動画配信の技術は様々あるが，ここでは HTTP を使用するものとそうでないものを大別する．近年では HTTP を使用する技術が増えてきており，その技術を使用したサービスが普及してきている．HTTP 使用するメリットは次のようなものである．まず，既存の WEB サーバが使用可能であるということ．次にファイアウォールに防がれにくいということである．HTTP 以外のプロトコルで規定された技術を使用する場合，それに応じたアプリケーショを別途用意する必要がある．そのため，多くのサーバに導入しようとするとかなりの導入コストがかかってしまうというデメリットがある．また，使用するネットワークによっては管理者によって特定のポートしか開かれていない場合もあり，ポートが閉じられているプロトコルが使用出来ない恐れがある．しかし，HTTP はほぼどのようなネットワークでも使用可能であるためネットワークによって使用出来ない恐れがほとんどないというメリットがある．そこで，ここではまず HTTP を使用しない技術を紹介し，次に HTTP を使用する技術を紹介する．そして最後に，HTTP を用いているがストリーミングの要素も併せ持つハイブリッドな技術について紹介する．

2.2.1　ストリーミング

ストリーミングはリアルタイム性の強い場面で利用されることが多い．前節で挙げたライブストリーミングがこのような場面に該当する．また，ここでは HTTP を使用しないものを挙げる．HTTP を使用したストリーミングは次々項で紹介する．

通常，ファイルはダウンロードを完了した後に開くが動画のような大容量のデータとなるとダウンロードにかかる時間が長く，リアルタイム性に大きな影響が出てしまう．そこで，ダウ

ンロードしながらの再生を可能にすることによってリアルタイム性を保つ技術がストリーミングである．また，動画の配送時間にシビアな技術であるためトランスポート層は UDP を使用する．さらにネットワークの制御が可能なプロトコルも存在する．ストリーミングを行うには従来の WEB サーバではなく，独自のプロトコルを用いたサーバアプリケーションが必要となる．特にストリーミング専用のプロトコルとして有名なものは，

- Microsoft Media Server (MMS)
- Real Time Streaming Protocol (RTSP)[8]
- Real Time Messaging Protocol (RTMP)

などがある．

2.2.2 プログレッシブダウンロード

プログレッシブダウンロードはストリーミングのように独自のプロトコルを用いず，HTTP を用いて動画を配信する技術である．ストリーミングと同様，ユーザは動画のダウンロードが完了する前にプレイバックを開始することが可能である．ストリーミングとの大きな違いは，動画データがエンドユーザのデバイスに残ることである．

プログレッシブダウンロードは HTTP を使用するために，サーバアプリケーションは従来の WEB サーバを使用することが可能である．このため，既存のサーバにデプロイするコストが低いというメリットがある．また，端末にキャッシュを残すためクライアントサイドでの高速なシークが可能となっている．しかし，ストリーミングと違い通信帯域の制御が困難なため通信速度が遅いとキャッシュが溜まるまでに時間がかかり，再生までに時間がかかるという問題がある．したがって，ストリーミングのようにライブ配信には向いておらず，ビデオ・オンデマンドに向いている．

プログレッシブダウンロードには著作権の問題もある．これは，動画をキャッシュするために後にユーザがローカルのファイルにアクセス可能であるという技術的な問題である．一方ストリーミングは再生後そのデータを破棄するように作られているプロトコルが多い．そのため，商用コンテンツを配信する課金サービスにおいてはストリーミングを採用することが多い．

2.2.3 Adaptive Bitrate Streaming

ストリーミングとプログレッシブダウンロードのハイブリッドのような存在が Adaptive Bitrate Streaming である．ストリーミングにおけるネットワークの制御が可能な点やプログレッシブダウンロードにおけるデプロイコストの低さなどが特徴である．また，DRM によって著作権保護が可能なプロトコルも多く存在する．主な例として，

- Apple HTTP Live Streaming (HLS)[9][10]
- MPEG Dynamic Adaptive Streaming over HTTP (DASH)[11][12][13]

- Adobe HTTP Dynamic Streaming (HDS)
- Microsoft Smooth Streaming

などがある．

Adaptive Bitrate Streaming はプログレッシブダウンロードと同様，HTTP を仕様している．HTTP はルータ, NAT, ファイアウォールの設定で制限されることがまずないために特に設定をすることなく，多くのネットワーク上のデバイスにコンテンツを配信することが可能となっている．さらに，HTTP は多くの Contents Delivery Network (CDN) がサポートしており，例えば Akamai のコンテンツ配信サービスである SOLA Sphere[14] は MPEG-DASH をサポートしている．また，研究分野でもコンテンツを置くサーバを複数用いた MPEG-DASH のスケールに関する研究などが存在する [15]．以上から，Adaptive Bitrate Streaming はスケーラビリティも高いと言える．

Adaptive Bitrate Streaming は，上で挙げたように様々なフォーマットがあるためコンテンツを全てのデバイスに対応させるのが困難である．さらに，これらのフォーマットのクライアント側での実装はハードウェアやオペレーティングシステムによって異なる．例えば Android ではバージョンによって HLS をサポートしているものとしていないものがある．そこで，プロトコルの数を減らして多種のデバイスへの配信を実現するために Adobe，Microsoft，Netflix などが集まって規格化しようとしたものが MPEG-DASH である．MPEG-DASH は ISO/IEC 23009-1[16] で定義されている．ここでは Adaptive Bitrate Streaming の仕組みを MPEG-DASH を例に説明する．

Adaptive Bitrate Streaming には動画視聴のシナリオとなる Index File が存在する．MPEG-DASH ではこのファイルを Media Presentation Description (MPD) といい xml で記述される．次に，動画データはいくつかのビットレートにエンコードされ，エンコードされた各データは 10 秒程度の segment と呼ばれるデータに分割される．MPD は DASH クライアントの挙動を指定するシナリオの他，用意されたビットレートに対応する動画の segment の URL などを記述している．こうして用意した MPD と segment に対して，一般に DASH クライアントは下記及び図 2.1 のように動作する．

1. MPD を取得する
2. MPD に指定された segment データを順にダウンロードし始める
3. segment をダウンロードする度にネットワークの帯域を測る
4. MPD でビットレートの変更が許可されていた場合, 3 に基づいて次に取得する segment のビットレートを変更する
5. 3,4 を繰り返す

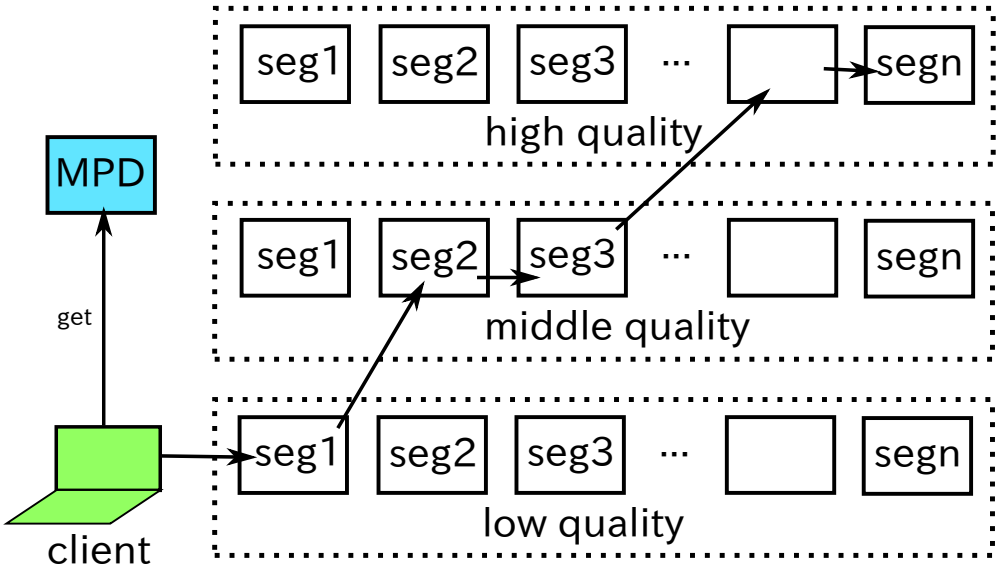

図 2.1. MPEG-DASH の動作

第 3 章

動画と SNS の連携

前章ではインターネット上で利用されている動画配信技術についての説明をした．それを踏まえ，本章では動画配信と SNS コンテンツを連携する技術を紹介する．また，動画と SNS を連携する際に発生する問題点を挙げ，その問題への対策例を説明する．

3.1 Hybridcast

Hybridcast とは，NHK が開発している放送通信連携システムであり，同報性・高品質・高信頼という放送の特徴と，視聴者の個別の要求に応えることができるという通信の特徴を生かしたシステムを構築するための仕組みである [17][18]．放送波によって送られた動画と IP ネットワークを通じて配送されたコンテンツを連携し，より豊かな放送サービスの実現を目指している．

放送波は遅延のない配送網であるが，IP ネットワークでの通信は遅延が発生する．そこで，Hybridcast は取得した動画のデータをバッファしておき，それと後から届く IP ネットワーク越しのコンテンツを同期するものである．放送波から取得した動画には mpeg-ts 内に NHK の時刻サーバの時刻情報を埋め込んでいる．また，IP ネットワーク越しのコンテンツも NHK の時刻サーバの時刻情報を付随させる．こうすることで配送された異なるメディアに同一源の時刻の情報が付随するためクライアントサイドで同期することが可能となっている．

この技術は，同期自体はクライアントサイドで行なっているが時刻情報はサーバサイドで担っており，完全にクライアントサイドで動作するものではない．したがって NHK というサービス提供者しか連携するメディアを選べないという問題がある．

3.2 連携において発生する問題

本項では IP ネットワークを通じて配送された動画と SNS コンテンツを連携する際に発生する問題を述べる．Hybridcast は動画を放送波を通じて配送するため，配送遅延がないものとして扱うことが出来た．そのため，IP ネットワークを通じて配送されたコンテンツの時間に動画を同期することでずれを回避した．逆に両者が IP ネットワークで配送される場合，ネッ

トワーク帯域によってデータサイズの大きい動画の配送に時間がかかる恐れがある．また，Twitter などの SNS コンテンツは基本的にテキストコンテンツを扱うため，動画の配送遅延に比べ遅延が無視できるほど小さいとみなすことが可能である．したがって，動画に SNS コンテンツを同期することでずれを回避することが可能となる．しかし，SNS コンテンツとの同期で一番大きな問題となるのは動画の配送遅延である．到着が遅延した動画を視聴した際にあるシーンについてのコメントを SNS 上で行った時，そのコメントは遅延がないユーザにとっては遅れてコメントが届いたように見える (図 3.1)．

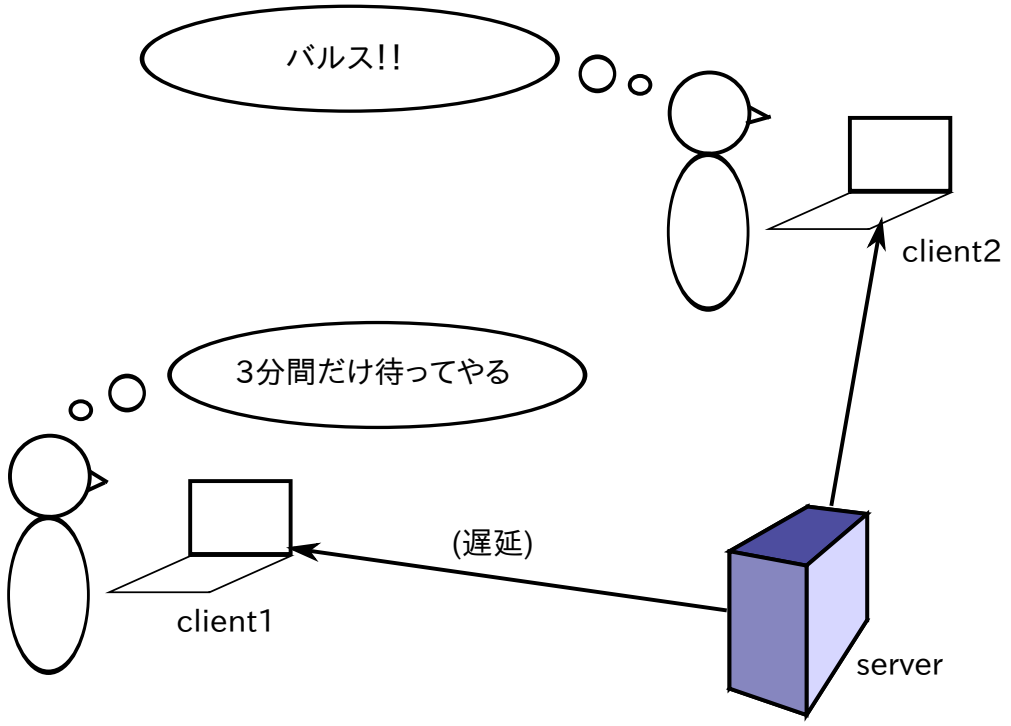

図 3.1. 動画遅延に起因するツイートの遅延の例

本研究では動画と SNS コンテンツの同期をクライアントサイドで行う．サーバサイドで同期を行う場合，例えばニコニコ動画のように各コメントに動画の再生位置を timestamp として付加することによって容易に同期をすることが可能である．予め同期されたデータが配送されるため，ユーザは動画とコメントのずれを感じることはない．しかし，クライアントサイドでの同期は同期する前のデータが遅延を伴って配送される．例えば YouTube Live を見ながらと Twitter を利用するといった場合がこれにあたる (図 3.2)．したがって，この配送の際に発生した遅延を考慮した同期の技術が必要となる．

動画と SNS コンテンツの同期では，動画が遅れる場合と SNS コンテンツが遅れる場合の 2 通りが考えられる．動画が遅れる場合とは，ネットワーク帯域が十分でないためにデータサイズが大きい動画を取得するのに時間がかかってしまい発生する状況を指す．この場合，他の動画の遅延がないユーザのコメントが表示されることによって番組のネタバレになるという問題がある．動画視聴の際におけるネタバレは視聴者にとってはかなり深刻な問題であり，SNS コ

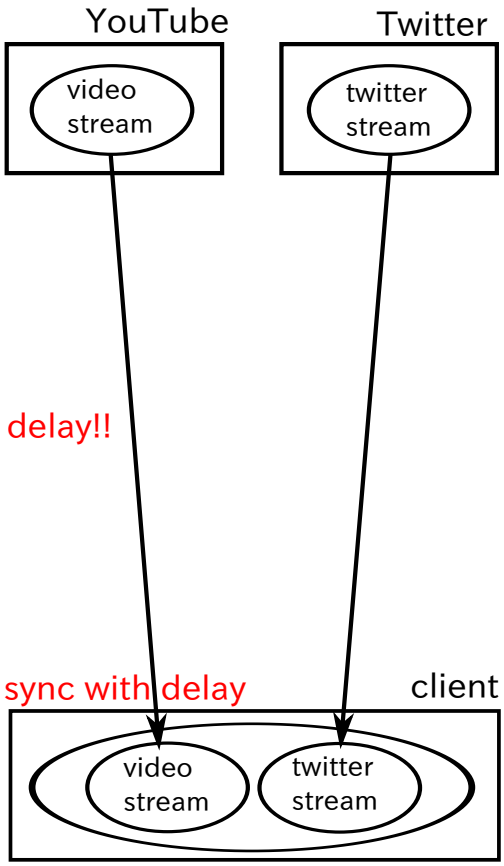

図 3.2. クライアントサイドにおけるメディアストリームの同期の例

ンテンツが遅れる場合に比べ対策する重要度が高い．SNS コンテンツが遅れる場合とは，他のユーザが動画を取得する際に遅延が発生しそのユーザが遅れて視聴したシーンに対してコメントをすることによって，自分にとってはある程度時間が経過したシーンについてのコメントが表示される状況を指す．このような場合，視聴者は他の SNS ユーザと一緒に動画を見ているはずなのに同時に見ているライブ感が得られないという問題がある．こちらは，ユーザ体験が悪くなるという問題があるが，ネタバレほど深刻な問題ではない．

3.3 問題への対策

前節で挙げた問題について，その対策となるサービスや技術が幾つか存在する．本節ではそのサービスや技術を述べる．

3.3.1 Twivo

Twivo とは，Twitter のツイートから番組のネタバレを含むものを取り除くアプリケーションである．2013 年の TVnext というハッカソンにおいて，唯一の女性参加者であった Jen Lamere がこのアプリケーションで優勝した．このアプリケーションは，自分がテレビ番組を

見られない状況で且つ Twitter を見ている場合，予め単語を登録しておくとその単語含まれるツイートを取り除いてくれるというものである．そして，自分が見られる状況になったら hit ボタンをクリックすることで取り除かれていたツイートが表示されるようになる．Chrome ウェブストアでアプリケーションが公開されている [19].

3.3.2　IEEE802.1AS

IEEE802.1AS[20] は IEEE1588[21] をベースにし，家電用途に仕様を簡単にしたものである．例えば，左右のスピーカーで同じタイミングで音を鳴らす場合や複数のモニタを繋げて 1 つの巨大モニタとして扱う場合，それぞれの機器において同じ時刻に音を鳴らしたり動画を再生する必要がある．このような複数端末間の時刻同期の仕組みを規定している．IEEE1588 は計測機器および制御機器用時刻同期プロトコルである．LAN 経由での高精度な時刻の配信方法について規定しており，非同期のネットワークを経由して，グランドマスター (マスター) のクロックをクライアント (スレーブ) のクロックへ時刻情報を正確に転送することを可能としている．

3.3.3　動画と SNS コンテンツの時間調整

本研究の目的である動画と SNS コンテンツのクライアントサイドでの同期について，本節で挙げた対策では解決出来ないことがわかる．Twivo はネタバレを防ぐことは出来るが，Twitter だけに限られており，また SNS コンテンツが遅れることによるライブ感を得られない問題に対しては有効でない．IEEE802.1AS は基本的に LAN 内での規格であるため SNS に所属している他のユーザとの同期やりとりに適していない．したがって，動画と SNS コンテンツの同期問題は次章で述べる提案手法により解決することとする．

第 4 章

提案手法

前章では動画と SNS コンテンツを連携するサービスと技術を紹介し，それらをクライアントサイドで同期することの問題点を述べた．本章ではその問題点を解決する手法を提案し，提案手法を実現するために必要な機能要件を整理する．そして，機能要件を満たす具体的な実装について説明する．

4.1 想定環境

想定環境は以下とする．

- SNS でつながりのあるユーザ群を 1 つのグループとみなす
- グループに所属するユーザが同じ動画を同じ時刻に視聴開始する
- 同じ端末で動画と SNS コンテンツを同時に表示する

まず SNS でつながりのあるユーザ群を 1 つのグループとみなし，そのグループに所属するユーザの間における動画と SNS コンテンツのずれを同期対象とする．本研究ではリアルタイム性を重視するため，Twitter を使用し，Twitter においてお互いにフォローしあっているユーザを 1 つのグループとみなす．つまり，図 4.1 のようにユーザ A がフォローしている全フォロワーの中でさらにユーザ A をフォローし返しているユーザとユーザ A を合わせたものを 1 つのグループとみなす．次に，グループに所属するユーザは同じ動画を同じ時刻に視聴開始するものとする．例えば 21:00 に放送開始される動画があった場合，そのグループのユーザは皆 21:00 に視聴開始し，その動画に対するコメントを Twitter に投稿する．動画が 10 秒遅れているユーザがあるシーンに対するツイートをすると，遅れがないユーザにとってはそのシーンの 10 秒後にツイートが表示されるはずである．また，これらの環境は各ユーザにおいて 1 つの端末上で行われるものとする．つまり，PC で動画を視聴しながらスマートフォンで Twitter を見るといった状況は考慮せず，1 つの PC 上で動画と Twitter を表示している状況を対象とする．

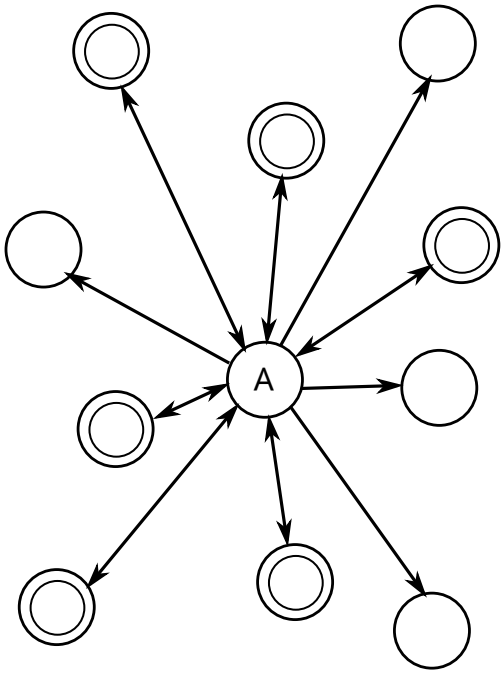

図 4.1. 想定環境におけるグループの定義

4.2　機能要件

前節では本研究で対象とする想定環境を述べた．本節では，その環境下で動画と Twitter のツイートを同期するために必要な機能要件を述べる．

まず，動画の再生位置に再生されるべき時刻情報が埋め込まれている必要がある．これは，動画とツイートを同期する際に，現在再生している動画の位置の遅延がなかった場合に再生されるはずの時刻を知るためである．この情報があることで，現在時刻と再生されるはずの時刻とのオフセットが動画の遅延時間と計算することが可能である．そのため，予め時刻情報を埋め込むことが可能な mpeg-ts を用いたストリーミングを使用するか，Index File に時刻の情報を記述することが可能な Apple HLS や MPEG-DASH を使用する必要がある．また，本研究では同期手法を独自実装するため，動画データの取得アルゴリズムを制御可能でなければならない．Apple HLS は取得アルゴリズムが Apple によって実装されたものを使用しなければならないため除外される．さらにユーザの導入コストを減らすため，新たにアプリケーションを導入する必要のない技術を用いるのが好ましい．MPEG-DASH は Chrome ブラウザ上で動作する上，ブラウザは普段使うものなので導入コストが低い．そこで，本研究では Chrome ブラウザ上で MPEG-DASH を使用することとする．

次に，SNS コンテンツの投稿時間がそのコンテンツに埋め込まれている必要がある．前節

で述べたように本研究ではリアルタイム性を重視するために Twitter を使用する．Twitter は各ツイートに投稿時間の情報が付随しており，Twitter API を使用して取得することで秒オーダーの時刻を取得することが可能であるため，この要件を満たす．

最後に，動画とツイートの同期，つまり再生位置調整機能が必要となる．前に挙げた動画とツイートの時刻情報はここでの同期において必要となる．ツイートはテキストデータであるため動画に比べて遅延が無視出来るとすると，遅れて到着した動画データに合わせてツイートを表示する必要がある．そのため，ツイートをバッファする機能が必要となる．また，動画はネットワーク帯域に応じて遅延が起きないような取得手法が必要である．

4.3 実装

本章では前節で述べた機能要件を満たすような実装について説明する．

4.3.1 クライアントサイド

機能要件で述べたように Chrome 上で MPEG-DASH を用いて動画を取得する．そのため，クライアントサイドは javascript で実装する．

まず，ネタバレ問題を解決する実装について説明する．これは 3.2 節で述べたように，自端末の動画取得に遅延が発生してしまった場合，グループに所属する他の遅延がない，もしくは自分より遅延の小さいユーザのツイートが動画の該当シーンよりも早く表示されてしまうことにより発生する問題である．この問題を解決するために，まず取得したツイートは全て一時的にバッファするようにする．次に，動画の再生位置がバッファされたツイートの時刻より遅れているか進んでいるかを判定する．動画の再生位置と現在時刻の関係は図 4.2 のようになる．また，動画の遅延がないユーザをユーザ B，遅延のあるユーザをユーザ A とする．動画の配信開始時刻を t_s，ツイートを行う動画の再生位置を Δt_{play} とし，時刻 $t_s + \Delta t_{play}$ にユーザ B がツイートをしたとする．またツイートを受け取る側であるユーザ A は Δt_{delay_A} の遅延が発生しているものとする．すると，ユーザ A にとっては動画の再生位置が $\Delta t_{play} - \Delta t_{delay\_A}$ のときにツイートがバッファされるため，Δt_{delay_A} だけ待ってからツイートを表示しなければならない．したがって，ユーザ B のツイートの投稿時刻を t_{tweet_B} とすると，ユーザ A は時刻

$$t_{tweet\_B} + \Delta t_{delay\_A} \tag{4.1}$$

となった時にツイートを表示すればよいことがわかる．この処理をバッファされたツイートに対して逐次行う．

次に動画の遅延自体を小さくする．上記のように自端末への動画遅延によって発生するネタバレはツイートをバッファすることで解決可能である．しかし，逆に遅延がないユーザにとっては遅延のあるユーザのツイートは動画のシーンに対して遅れて表示されるため，ライブ感が得られない．さらに，バッファによるネタバレ回避はネタバレを防ぐことは可能だが，遅延がないとみなしていたユーザにも動画の配送遅延があった場合，その遅延分が時間調整した部分

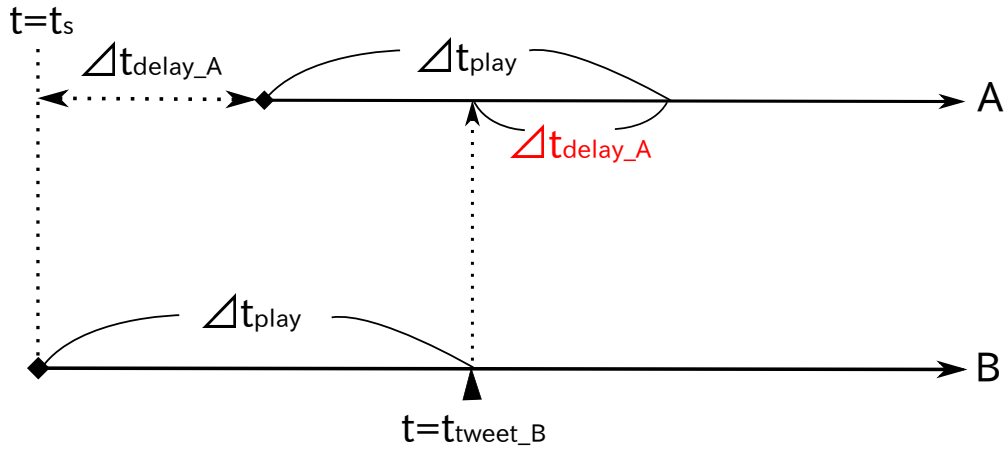

図 4.2. 動画の再生位置と現在時刻の関係 (A)

よりもさらに遅れて表示されるという問題がある (図 4.3)．つまり，動画の配送遅延がある限りライブ感が欠落するという問題は解決出来ない．

ここで図 4.3 から，ネタバレ防止手法を使用した場合のユーザ A，B のシーンに対するツイートの遅れがどうなるかを求める．この図は時刻 t_s に配信開始される動画に対して，Δt_{play} 後にそのシーンのツイートをすることを示している．またユーザ A の動画遅延を Δt_{delay_A}，ユーザ B の動画遅延を Δt_{delay_B} とする．

まず，ユーザ A でのシーンに対するツイートの遅れを求める．ユーザ A はネタバレ防止手法により B のツイートを再生位置 $\Delta t_{delay\_B} + \Delta t_{play}$ のものとみなすため，自分の再生位置がその時間になったときにツイートを表示する．したがって，時刻

$$\begin{aligned} & t_s + \Delta t_{delay\_A} + (\Delta t_{delay\_B} + \Delta t_{play}) \\ = & (t_s + \Delta t_{delay\_A} + \Delta t_{play}) + \Delta t_{delay\_B} \\ = & \ t_{tweet\_A} + \Delta t_{delay\_B} \end{aligned} \tag{4.2}$$

にユーザ B のツイートを表示する．つまり，Δt_{delay_B} だけシーンよりツイートが遅れることとなる．

次に，ユーザ B でのシーンに対するツイートの遅れを求める．こちらも同様にユーザ A のツイートを再生位置 $\Delta t_{delay\_A} + \Delta t_{play}$ のものとみなすため，自分の再生位置がその時間になったときにツイートを表示する．したがって，時刻

$$\begin{aligned} & t_s + \Delta t_{delay\_B} + (\Delta t_{delay\_A} + \Delta t_{play}) \\ = & (t_s + \Delta t_{delay\_B} + \Delta t_{play}) + \Delta t_{delay\_A} \\ = & \ t_{tweet\_B} + \Delta t_{delay\_A} \end{aligned} \tag{4.3}$$

にユーザ A のツイートを表示する．つまり，Δt_{delay_B} だけシーンよりツイートが遅れることとなる．

上記から，ネタバレ防止手法を適用後は動画の遅延がそのままシーンに対する遅延になることがわかる．この遅れを解決するために，MPEG-DASH の Adaptive Bitrate，つまり帯域に

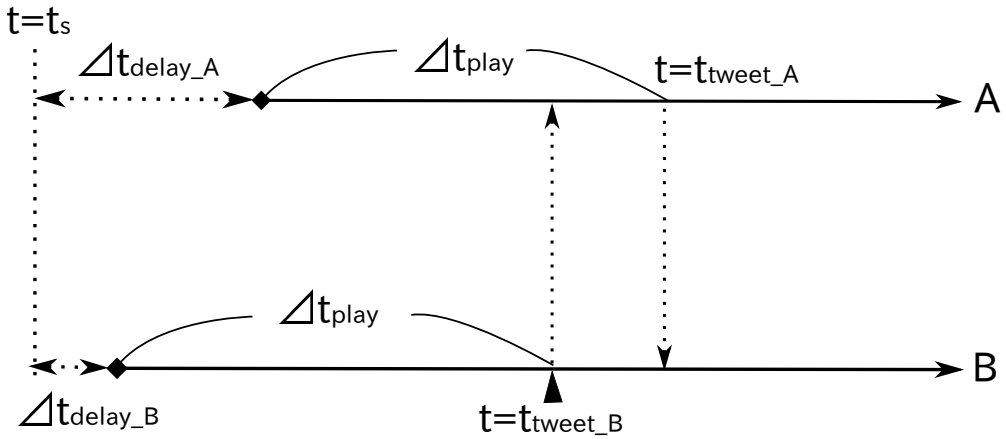

図 4.3. 動画の再生位置と現在時刻の関係 (B)

適応したビットレートの動画を選択することで動画の取得時間による遅れをなくすことを目指す．第 2 章で説明したように，MPEG-DASH では 1 つの動画コンテンツを複数のビットレートにエンコードしたストリームを用意し，クライアントは帯域に応じたものを選択するという手法を用いている．この手法を用いることで遅延のない動画視聴をすることはある程度は可能であるが，segment のデータサイズや duration を考慮しておらず，また不安定なネットワーク環境においてバッファリングが発生する恐れがある．本研究では動画の配送遅延が最も深刻な問題となるため，動画の画質を犠牲にしてでも遅延がないストリームをうまく選択出来るようなものにしたい．そこで次のようなプロセスを行うことでこれを実現する．

1. 帯域推定
   前の segment のデータ取得時間とデータサイズから現在のネットワーク帯域を推定する．帯域推定方法は既存実装の DASH-JS や webm-DASH のアルゴリズムを参考にした．
2. HEAD リクエスト
   ストリームにおける次に取得する segment データに対し，HEAD リクエストを送信する．このリクエストに対するレスポンスから該当 segment のデータサイズを取得する．ループの初回は最高品質のストリームのデータを対象とする．
3. segment 取得にかかる時間の推定
   segment のデータサイズを $datasize$(byte)，ネットワーク帯域を $bandwidth$(bps) とすると，segment を取得するのにかかる時間 t_{GET} は
   $$t_{GET} = \frac{datasize \times 8}{bandwidth} \tag{4.4}$$
   で推定することが出来る．
4. ストリームの再選択
   バッファされている時間と t_{GET} を比較する．バッファされている時間よりも取得にかかる時間の方が長い場合，必ずバッファリングが起きて動画が一時停止してしまい遅

延に繋がる．したがって，1 つビットレートの低いストリームに選択し直す．ただし，バッファに対して取得時間のかかる割合が大きいと次の segment を取得するまでの時間がなくなってしまうため，t_{GET} に 0.8 倍の重み付けをして計算する．

5. 処理の繰り返し
   4 でバッファされている時間よりも取得にかかる時間の方が短くなるまで 2〜3 の処理を繰り返す．

4.3.2　サーバサイド

サーバサイドでは，前項で述べたクライアントサイドの実装を満足するように動画を用意し，WEB サーバの設定も行なっておく．

まず，帯域に合った segment の取得をするため，及び取得時間がかからないようにするために，通常のネットワーク帯域よりもかなり低いものから動画のオリジナルのビットレートまでに渡る幅広い画質にエンコードしておく．低いビットレートの目安は 3G 回線の平均帯域とする．次に，WEB サーバに cross-domain access を許可する設定をする．これは，MPEG-DASH などの Adaptive Bitrate Streaming 技術では，ネットワーク帯域に応じてデータを取得するサーバの変更も動的に可能であることから，WEB ページのドメインと動画を取得しに行く先のドメインが異なる場合があるためである．MPEG-DASH では MPD や segment を XMLHttpRequest によって取得する．しかし，XMLHttpRequest は Same Origin Policy というセキュリティ上のポリシーにより，URL と異なるドメインへのアクセスが出来ない．これを解決するために作られた仕様が Cross-Origin Resource Sharing (CORS)[22] である．この仕様に基づいてやりとりを行えば cross-domain access が可能となる．

第 5 章

実験・評価・考察

前章では動画と Twitter をクライアントサイドで同期するための手法を示した．本章ではそれをもとに，実際に実験，評価，考察を行う。

5.1 実験

本節では，前章で提案した手法を用いて実際に動画と Twitter を同時に再生し，その際に発生するずれを測定する．そして，提案手法を用いた場合とそうでない場合のオフセットを求める．

5.1.1 実験環境

前章の想定環境では Twitter の相互フォローの関係にあるユーザ群を 1 つのグループとみなした．そのような環境を簡略化すると，グループ内の自分以外のユーザを 1 人抽出し，そのユーザと自分の 2 人で測定を繰り返し行えばよい．したがって，この実験ではユーザ A(自分) とユーザ B(その他のユーザ代表) の 2 人で比較を行う．また，ユーザ A への動画配信遅延の方がユーザ B へのそれよりも遅いとしても一般性は失われない．したがって，図 4.3 のようにあるシーンのツイートについて，ユーザ B のツイートはユーザ A にはシーンより早く到着し，ユーザ A のツイートはユーザ B にはシーンより遅く到着する環境を想定する．

サーバに用意する動画のビットレートは

- 500kbps
- 1Mbps
- 2Mbps
- 4Mbps
- 8Mbps
- 16Mbps
- 22Mbps

を用意した．また，各 segment の duration は 5 秒としている．使用したマシンのスペックは表 5.1 である．

	サーバ
ハードウェア	デスクトップ
CPU	Intel Core i7-3770K CPU @ 3.50GHz
メモリ	32GB
OS	Ubuntu 13.04 raring
Ethernet Controller	Realtek RTL8111/8168

	端末 1(ユーザ A)
ハードウェア	ThinkPad X201i
CPU	Intel Core i3 CPU M 380 @ 2.53GHz
メモリ	4GB
OS	Ubuntu 13.10 saucy
Ethernet Controller	Intel 82577LM
Network Controller	Intel Centrino Advanced-N

	端末 2(ユーザ B)
ハードウェア	ThinkPad X200s
CPU	Intel Core2 Duo CPU L9600 @ 2.13GHz
メモリ	4GB
OS	Ubuntu 13.10 saucy
Ethernet Controller	Intel 82567LM
Network Controller	Intel Ultimate N WiFi Link 5300

表 5.1. 実験に使用したマシンのスペック

5.1.2　ユーザ A へのネタバレを防ぐ実験

実験環境では，ユーザ A へ動画のシーンより早くそのシーンのツイートが届いてしまうため，ネタバレが発生してしまう．そこで，本項では提案手法を適用した場合と適用しない場合で，ユーザ A の端末に実際にシーンが再生される時間とツイートが表示される時間を比較する．図 5.1 のようにシーンに対するツイートを表示する時刻を t_{tweet}，動画の該当するシーンが再生される時刻を t_{play} とし，2 つのオフセット t_{offset} を

$$t_{offset} = t_{tweet} - t_{play} \tag{5.1}$$

と定義すると，

$$\begin{cases} t_{offset} \geq 0 : \text{ネタバレにならない} \\ t_{offset} < 0 : \text{ネタバレになる} \end{cases} \tag{5.2}$$

となる．提案手法を適用するとこの t_{tweet} の値が変わるため，t_{offset} の値も変わる．そこで，提案手法を適用した場合としない場合でこの t_{offset} の値を測定する．この実験では約 10 分の動画を端末 1 とサーバで再生し，動画が 1 分になった時点でツイートをする．ユーザ A の動画の遅延を発生させるために帯域制限をし，またユーザ B については動画取得時間を 0 としたいのでサーバ上で動画再生及び Twitter の投稿を行った．測定した結果は表 5.2 のようになった．

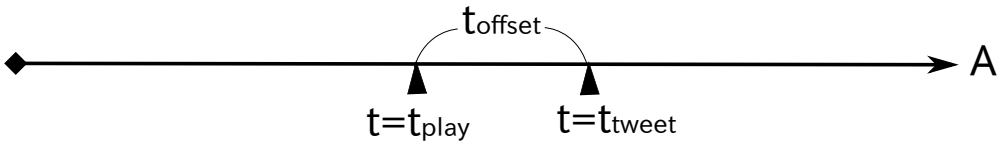

図 5.1. ネタバレ防止実験の環境

	提案手法適用前	提案手法適用後
平均 t_{offset}	-16.1825	0.7278
ネタバレ発生率	1.0	0.0

表 5.2. ネタバレ防止実験の測定結果

5.1.3 動画の遅延を小さくする実験

前項ではユーザ B の動画遅延を 0 として実験した．しかし，実際にはユーザ B の遅延も考慮しなければならない．ネタバレ防止手法を適用し，ユーザ B の遅延も考慮すると，前章より，ユーザ A，B にとって Δt_{delay_B}，Δt_{delay_A} のツイートの遅れが生じる．そこで，動画の配送時間を小さくすることで解決する．

前項と同様に，提案手法を適用した場合としない場合の動画取得時間を比較する．まずは，同端末において動画の遅延が提案手法によってどれだけ変化するかを調べる．提案手法適用前の segment を選択するアルゴリズムは既存実装の DASH-JS や webm-DASH の実装を参考にした．計測する対象の項目は，初回バッファリング遅延，再生位置 1 分での遅延，総遅延，選択した segment の平均 bitrate である．

ネットワーク環境が有線 (帯域制限なし)，無線 (bld2-guest)，有線 (無線の帯域と同じ帯域に制限) の場合において，これら値を測定した結果は表 5.3 のようになった．

またこれらのうち，遅延が起きている無線環境と，帯域制限を行った有線環境における

- buffer: バッファ [sec]
- interrupt: 遅延が発生した時間

有線環境での測定結果: iperf 173Mbps

	提案手法適用前	提案手法適用後
初回バッファリング遅延 [sec]	0.792	0.113
再生位置 1 分での遅延 [sec]	0.792	0.113
総遅延 [sec]	0.792	0.113
動画平均 bitrate[Mbps]	22.609	20.472

無線環境 (bld2-guest) での測定結果: iperf 4.58Mbps

	提案手法適用前	提案手法適用後
初回バッファリング遅延 [sec]	0.092	0.148
再生位置 1 分での遅延 [sec]	7.509	0.148
総遅延 [sec]	15.718	0.148
動画平均 bitrate[Mbps]	2.549	4.140

有線環境 (帯域制限) での測定結果: iperf 4.45Mbps

	提案手法適用前	提案手法適用後
初回バッファリング遅延 [sec]	0.195	0.124
再生位置 1 分での遅延 [sec]	1.753	0.124
総遅延 [sec]	11.775	0.124
動画平均 bitrate[Mbps]	4.087	5.083

表 5.3. 動画遅延削減実験の測定結果

- throughput: 推定帯域 [kbps]
- bitrate: 選択した segment の bitrate[kbps]

も同時に測定したところ，図 5.2 - 図 5.9 のような結果が得られた.

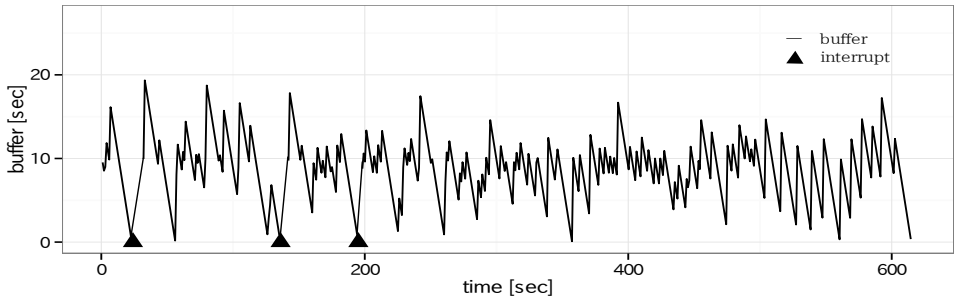

図 5.2. 提案手法適用前の無線環境実験測定結果 (A)

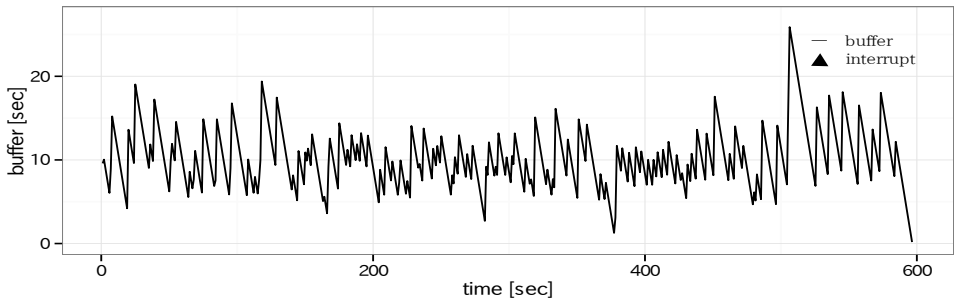

図 5.3. 提案手法適用後の無線環境実験測定結果 (A)

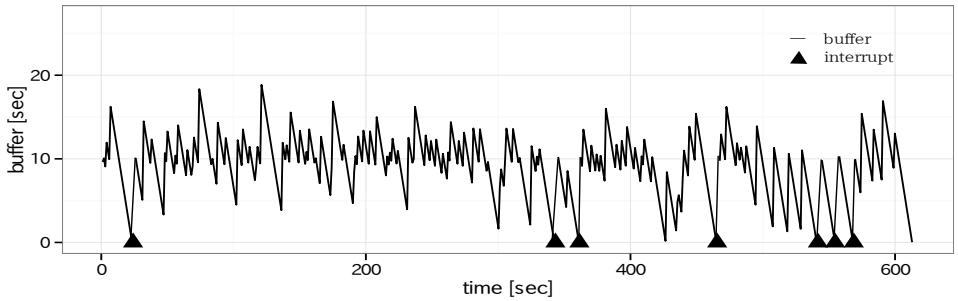

図 5.4. 提案手法適用前の帯域制限された有線環境実験測定結果 (A)

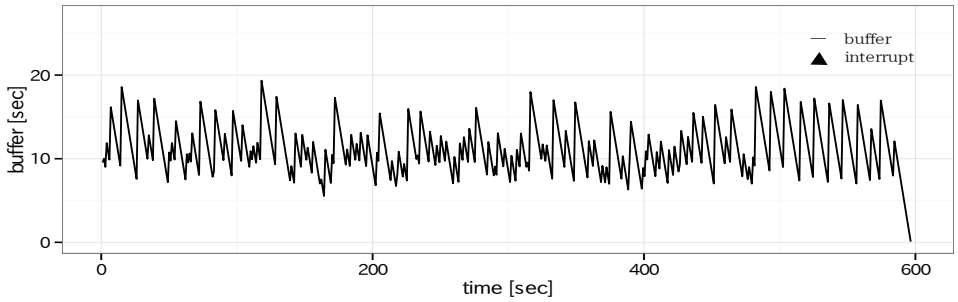

図 5.5. 提案手法適用後の帯域制限された有線環境実験測定結果 (A)

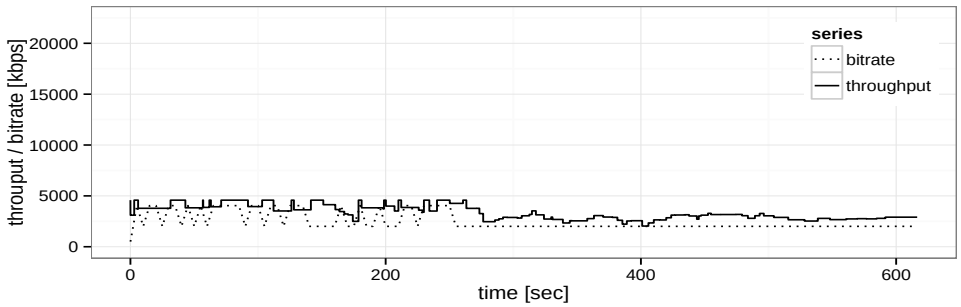

図 5.6. 提案手法適用前の無線環境実験測定結果 (B)

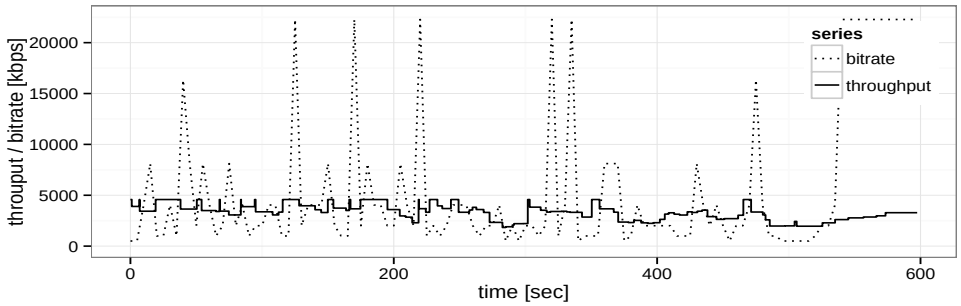

図 5.7. 提案手法適用後の無線環境実験測定結果 (B)

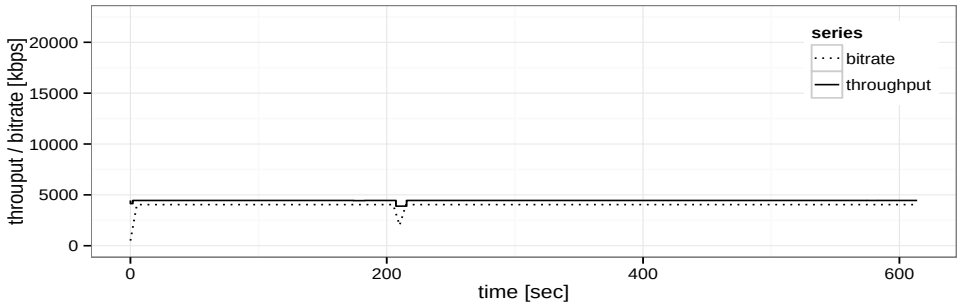

図 5.8. 提案手法適用前の帯域制限された有線環境実験測定結果 (B)

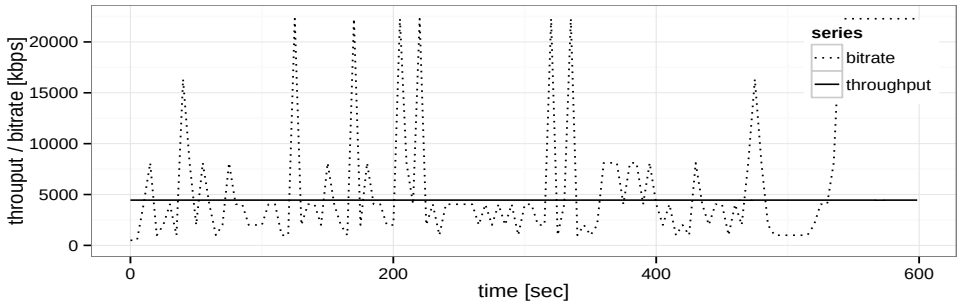

図 5.9. 提案手法適用後の帯域制限された有線環境実験測定結果 (B)

次に，複数端末においての比較をする．この実験ではネタバレ防止手法と動画の遅延少なくする手法を両方適用して Twitter のツイートの遅れを測定する．まず，ユーザ A，B 両者が同一時刻に動画を再生開始し，動画の再生位置が 1 分となったところで Twitter でツイートを行う．その後，相手のツイートが表示される時刻から動画が 1 分になった時刻を引いたもの，つまり動画のシーンに対するツイートの遅れを求める．まずユーザ A，B ともに有線環境で行い，次にユーザ A は無線 (ist-members)，ユーザ B は無線 (bld2-guest) で実験を行った．得られた結果は表 5.4 のようになった．

	ユーザ A の平均ツイート遅れ	ユーザ B の平均ツイート遅れ	両者の平均
有線環境	1.2797	1.0274	1.1534
無線環境	0.7222	1.2467	0.9845

表 5.4. 動画 1 分後のシーンに対するツイートの遅れ (秒)

5.2 評価

本節では，前節で得られた実験結果の評価を行う．

5.2.1 ネタバレ防止実験の評価

ネタバレを防止する実験では，提案手法が正常に機能すれば理論的に 100% ネタバレは起きないと予想できる．実験結果である表 5.2 より，100% ネタバレが発生するようなネットワーク環境においても提案手法を適用すればネタバレを完全に防ぐことが可能であることがわかった．したがって，本手法はネタバレを回避するという点で有効であるということがわかる．

5.2.2 動画の遅延を減らす実験の評価 (1)

先に動画の遅延を減らす手法を適用することでどの程度の遅延が改善可能であるかを評価する．実験結果は表 5.3 である．

まず有線環境の結果を評価する．有線環境では提案手法を適用した場合としない場合で遅延に関する差は見られなかった．これは，用意した動画のストリームの品質に対してネットワークの帯域が十分に確保出来ているからだと考えられる．実際，最高品質のストリームは 22Mbps であるのに対し，iperf の結果は 173Mbps であった．動画の平均 bitrate は，前者が 22.609Mbps で後者が 20.472Mbps であった．こちらは提案手法を適用しない方が良い結果が得られている．提案手法は動画の品質を犠牲に再生中のバッファリングが起きないようにしようとするものである．実際のネットワーク帯域は動画の品質に対して十分であったが，提案手法のアルゴリズムが働くことで品質を落としたストリームを選択している．したがって，ネットワークの帯域が十分である場合，提案手法を適用すると低いストリームが選択されてしまう場合があることがわかる．

次に無線環境での結果について評価する．無線環境では，動画の再生位置が 1 分において提案手法適用前の遅延は 15.718 秒，提案手法適用後の遅延は 0.148 秒であった．最終的に総遅延は 15.718 秒と 0.148 秒となり，提案手法適用前では遅延が発生し，提案手法適用後では再生中のバッファリングが起きなかったことがわかる．これは，提案手法を適用した場合には動画のバッファが少なくならないように働くためにバッファリングが起きなかったためである．図 5.2，図 5.3 を見ると，確かに提案手法適用前では 3 回動画が停止しているが提案手法適用後は動画の停止がなく最後まで再生出来ている．つまり，提案手法を適用しない場合はストリー

ムの品質と推定したネットワーク帯域によって次 segment の選択を行なっているため，バッファが少なくなってしまっていることがわかる．また，提案手法適用前のバッファは変動が大きく少なくなることが多いのに対して，提案手法適用後はバッファの変動も小さく 10 秒前後を推移して安定していることがわかる．動画の平均 bitrate は前者が 2.549Mbps で後者が 4.140Mbps である．また，図 5.6，図 5.7 より，提案手法適用前は推定したネットワーク帯域よりも品質が低いストリームしか選択していないのに対し，提案手法適用後はそれよりも高いストリームを選択していることがわかる．これは，提案手法が推定したネットワークの帯域だけでなく，バッファの残り時間及び次 segment のデータサイズも考慮したストリームの選択を行なっているため，品質の高いストリームを選択可能な場面ではそれを選択しているからである．つまり，提案手法によって視聴可能な動画の品質も高くなる傾向にあることがわかる．

最後に無線環境と同じネットワーク帯域に帯域制限した有線環境での実験について評価する．表からわかるように，遅延時間の差はあるが結果は無線環境と同じようなものとなった．ただし，図 5.8，図 5.9 から推定している帯域が無線環境よりも安定していることがわかる．これは，帯域制限したネットワークが有線だからであって，逆に無線環境のネットワークは不安定であるために推定帯域も不安定になるということを示している．図 5.4，図 5.5 からも同様のことが言える．これから，無線のような不安定なネットワーク環境で動画視聴を行った場合，帯域推定の精度が落ちることで次 segment のストリーム選択が最適でなくなる恐れがあることがわかる．本実験では提案手法適用後では動画再生中の停止は起きなかったが，さらに不安定なネットワーク環境では動画が停止してしまう恐れもある．

5.2.3　動画の遅延を減らす実験の評価 (2)

前項では提案手法を用いることで，安定したネットワーク環境であれば動画が再生途中で停止し遅延が発生することを防ぐことが可能であることがわかった．本項では，その手法とネタバレ防止の手法を共に適用することで Twitter のツイートがどの程度の遅れで表示されるかについての評価を行う．

実験結果は表 5.4 である．有線環境ではツイートの遅れの平均が 1.1534 秒，無線環境ではツイートの遅れの平均が 0.9845 秒であり無線環境の方が有線環境よりも遅れが小さくなっている．ここで，表 5.3 の提案手法適用後の遅延を考える．有線，無線で動画の遅延は 0.113 秒，0.148 秒となっている．また，4.3.1 項より動画のシーンに対するツイートの遅れは動画の遅延と同じになる．つまり，この実験結果で得られた値は表 5.3 の遅延に近くなると考えられるが，実際にはそれよりも約 1 秒の遅れが発生している．これは，本研究では動画はテキストに比べデータサイズが非常に大きいため，前者の遅延に比べテキストの遅延は無視出来るという仮定のもとで実験を行なっていることに起因する．つまり，提案手法を適用することで動画の遅れよりも Twitter の遅れの方が大きいことにより動画のシーンに対するツイートの遅れが予想した値よりも大きくなっていると考えられる．

5.3 考察

前節では実験結果についての評価を行った．この評価より，本節では本研究の応用例について考察する．また，動画のシーンに対するツイートの遅れを定量的に評価する方法について考察する．

5.3.1 ネタバレを防ぐ実験の考察

前節から，ネタバレを防ぐ機能を実装することで 100% ネタバレを防ぐことが可能であることをがわかった．一方で，この機能の特性上，応用としてニコニコ生放送のタイムシフト再生のような視聴の仕方も可能であることがわかる．これは，動画の配信時刻よりも遅れて視聴を開始した場合でも，Twitter のグループに所属する人のツイートが動画に合わせて表示されるものである．例えば 21:00 に配信される動画が合った場合，21:30 から再生を開始すれば 30 分遅れているのにツイートが 30 分遅れて表示されるため，擬似的にグループの他のユーザと一緒に動画を見ているような体験が得られる．

慶応義塾大学政策メディア研究科村井純研究グループでは，JGN[23] のテストベッドを用いて 4K 動画コンテンツとソーシャルメディアの連携を行うプロジェクトを行なっている．このプロジェクトは，LAN 内にある複数端末がクロックを同期することで，全ての端末での動画や SNS コンテンツの同期およびタイムシフトを可能にするものである．本研究のネタバレを防ぐ手法とユーザ体験は似ているが，本研究は LAN 内のみではなくグローバルな範囲まで適用可能であるという差がある．本研究は範囲を広げることで村井純研究グループのプロジェクトに比べ同期精度を犠牲にしている．

5.3.2 動画の遅延を減らす実験の考察

本項では 5.1.3 項の結果についての考察を行う．

遅延が発生する原因

提案手法を適用しない場合において，なぜバッファが足りなくなり遅延が発生するのかを考える．

MPEG-DASH では動画を複数の異なる品質，つまりビットレートにエンコードした動画を用意し，それをある時間に区切ってストリームとする．また，動画は基本的にエンコードの際のコーデックは h264 で行われる．エンコードされたデータはフレームによって動きが激しい時とそうでないときがある．この際，動きが激しい部分に比べ動きがない部分の圧縮効率は高くなる．MPEG-DASH で用意する区切られたデータ，つまり segment は，等間隔で区切るため，時間によってデータサイズが変化する．実際に本研究で使用した動画においても，8.4M バイトの segment の次が 80M バイトの segment になっている部分が存在する．既存手法のように，現在のネットワーク帯域と動画のビットレートの比較においてストリームの品質を選

択しようとする場合，帯域が一定ならば 8.4M バイトの segment も 80M バイトの segment も同じように評価される．しかし純粋に 10 倍程度のデータサイズの差があるため，当然取得にかかる時間もそれに近い差が出ることとなり，バッファされている時間を超えて停止してしまうことが起きる．本実験で提案手法を適用する前に起きていた停止はこれに起因する．

Twitter の実データとの比較

本研究の動画の遅延を減らす手法を用いた場合，動画のシーンに対するツイートの遅れは表 5.4 の値が得られた．しかし，その値は実際の Twitter においてどの程度の遅れに分布しているのかがわからない．そこで本項では，実際の Twitter のツイートの分布を元に，実験結果で得られた遅延と改善度を定量的に評価する．

動画のシーンに対するツイートというのは，あるイベントに誘発されるツイートに一般化される．具体的にはスタジオジブリのアニメーションである天空の城ラピュタにおける「バルス」というツイートや，年越しの際にツイートされる「あけおめ」というツイートである．このようなイベントに誘発されるツイートはイベント発生時刻とツイートの時刻の差が非常に小さく，また多くの視聴者がツイートを行なっているため，評価指標として優れていると考えられる．そこでここでは 2013 年 8 月 2 日に放送された天空の城ラピュタにおいて，23 時 22 分頃にアニメーション内で「「バルス」」という台詞が言われた時の Twitter のツイートをもとに実験結果について考察する．

実際に Twitter から Search API を使用して得られたデータは図 5.10 となる．Search API は全てのツイートを取得するわけではないので，全体のツイート数は実際の値と一致しない．2013 年 8 月 2 日 23 時 21 分 48 秒にテレビにおいて「「バルス」」と言われるシーンになり [24]，その 2 秒後に秒間あたりのツイート数が最高を記録した [25]．この記録は Twitter Japan によって公式に報告されている．図の time が 0 となっているのが 48 秒で，グラフが最高地点に達しているのが 2 秒後の 50 秒に対応する．横軸は前後 1 分の範囲を示している．また，赤く塗りつぶされている範囲は 0 秒から表 5.4 において最も遅れが大きかった値である 1.2797 秒後までの範囲である．このデータにおいて，0 秒から 1 分後までのツイートにおける 1.2797 秒後までのツイートの割合を求める．Twitter のツイートのタイムスタンプは秒単位なので，n 秒におけるツイート数を $N(n)$ とすると，1.2797 秒後までのツイート数は

$$N(0) + N(1) + 0.2797 \times N(2) \tag{5.3}$$

とする．この値は 206.8783 である．また 0 秒から 1 分までのツイート数の総和は 862 であるため，実験結果で得られた時間には約 24% のツイートが含まれることとなる．つまり，残り 76% のツイートよりは早くツイートを行うことが出来ていると言える．この図のデータは，テレビ放送という遅延のないメディアを通じて配信した動画を視聴してツイートしたデータであるため，そのデータにおけるイベント後 1 分間のツイートのうち 76% よりも早くツイートが出来ていることは十分遅れがないと考えられる．

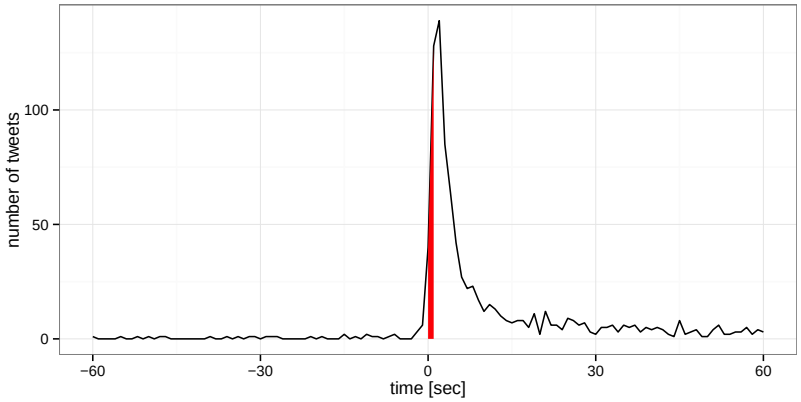

図 5.10. 天空の城ラピュタの「「バルス」」に誘発される Twitter のツイート

動画の高速再生を用いた遅延からの復帰

図 5.2 のようなネットワークが不安定な状況では動画が停止してしまう恐れがあることがわかるが，劣悪なネットワーク環境によっては提案手法を適用しても停止してしまう恐れがあると考えられる．そこで本項では万一遅延が発生してしまってもそこから復帰する方法について考える．

近年のテレビや PC の動画プレーヤーには高速再生機能が標準で存在している．そこで，遅延が発生してしまった場合は遅延がなくなるまで高速再生を行うことによって遅延から復帰することが可能であると考えられる．ただし，遅延があまりに大きい場合や，segment の取得時間があまりに大きい場合は高速再生を行うことで動画の停止をさらに頻発させてしまう恐れがある．したがって，ここでは 5.1.3 項で得られたバッファの情報が最大約 20 秒であることから，遅延が 20 秒以内のときに高速再生をして復帰を試みる．遅延が 20 秒を超えるような不安定であったり帯域が極端に低い場合は復帰を諦める．また，実験の際には遅延が発生するように 5.1.3 項で使用した手法は適用せずに行う．この手法を用いて 5.1.3 項と同様の実験を行った結果は表 5.5，図 5.11，図 5.12 である．

有線環境 (帯域制限) での測定結果: iperf 4.24Mbps

初回バッファリング遅延 [sec]	0.077
再生位置 1 分での遅延 [sec]	0.077
総遅延 [sec]	20.4
動画平均 bitrate[Mbps]	4.106

表 5.5. 高速再生機能による遅延削減実験の測定結果

表 5.5 の総遅延 20.4 秒は動画再生中に動画が停止して動画の再生が遅れた時間である．この値と，動画長の 596 秒との和である 616.4 秒が動画の再生を終えるはずの時間である．しかし，この実験では動画の遅延が発生した際に高速再生をすることで遅延がない時間に復帰する

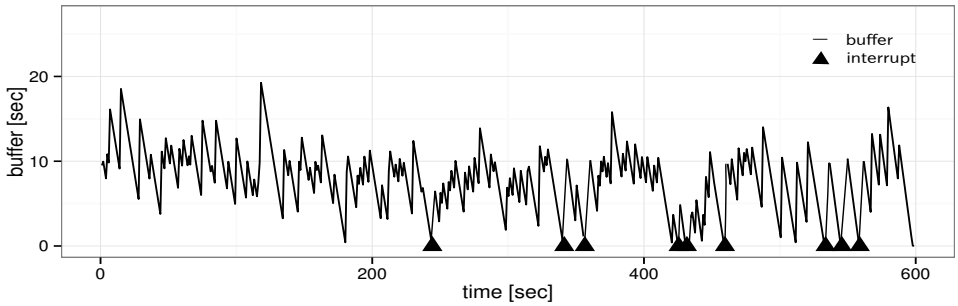

図 5.11. 高速再生機能による遅延削減実験の測定結果:バッファ

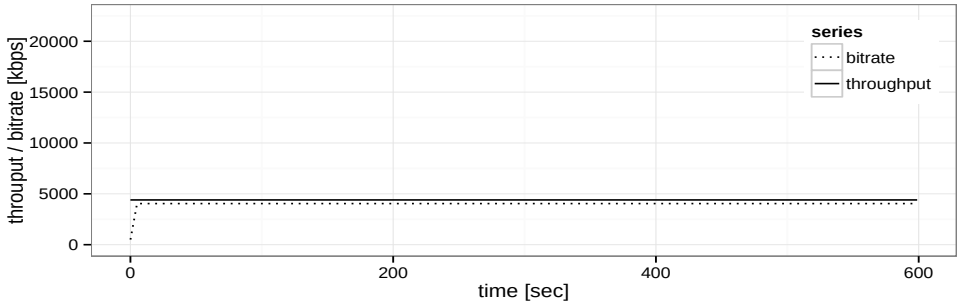

図 5.12. 高速再生機能による遅延削減実験の測定結果:スループット

機能を実装しているため，616.4 秒よりも早く再生が終了すると考えられる．ここで，図 5.11 を見てみると，動画の再生終了時刻が 616.4 秒よりも早くなっていることがわかる．具体的な数値は 558.824 秒であった．つまり，20 秒の遅延を 3 秒弱の遅延にまで減らしていることがわかる．したがって本手法は，動画の再生中にバッファリングが起き遅延が発生してしまった場合でも，元の遅延がない状況に復帰するのに有効であると言える．また，実際に動画を視聴していた際には，再生測度が変わったことを体感出来ない程度の変化であった．つまり，視聴者に動画が遅れていると気づかれてしまう恐れも減ることとなる．

ここで，動画の高速再生によって遅延から復帰することが可能であることがわかったが，動画の重要なシーンで高速再生しているとユーザ体験が下がる恐れがある．CinemaGazer[26] は字幕付きの動画において，字幕のない部分を高速再生することによって時間がないときでも動画を高速に視聴することを可能にしている技術である．このように，ユーザにとって重要度の低いと考えられる部分になったときに高速再生を行い遅延から復帰するということも考えられる．

第 6 章

結論

本章では，本論文のまとめと今後の課題について述べる．

6.1 まとめ

本研究では，インターネットの普及によって数多く出現してきたメディアに対し，複数のメディアストリームの再生タイミングをクライアントサイドで同期する手法を提案し，実装と評価を行った．同期するメディアストリームは，時間に敏感なメディアである動画と Twitter を使用し，動画の配信において動画の取得時間に起因する遅延をなくすために Adaptive Bitrate Streaming である MPEG-DASH を使用した．クライアントサイドの同期においては，Twitter による動画のネタバレを防ぐ機能と動画の遅延を小さくする機能を実装した．ネタバレを防ぐ機能は完全にネタバレを防ぐことが可能であることを示し，動画の遅延を小さくする機能は遅延の発生を少なくすることが可能であることを示した．また，これらの機能の併用によって動画のシーンに対する他のユーザが行ったツイートの遅れを実際の Twitter においてイベントに起因するツイートの 76% よりも早く表示することが可能であることを示した．

6.2 今後の課題

本研究では，複数のメディアストリームの再生タイミングをクライアントサイドで同期する技術を提案，実装した．提案した手法を適用しない場合，動画が再生途中にバッファが足りなくなりバッファリングして停止し，遅延となってしまうようなネットワーク環境でも，提案手法を適用すればバッファリングが起きないことを示した．しかし，動画の品質が FullHD のものを使用しており，さらに画質の高い動画を使用した場合でも同様の結果になるかがわからない．4K/8K に対応したテレビやコンテンツが登場し始めているため，このような高品質な動画を使用した際の挙動も調べる必要がある．

また，本研究では動画と Twitter の連携において，相互フォローしている人から 1 人代表を抽出して同期実験を行うという手法を用いた．実際の環境では複数人の相互フォローの関係にある人からなるグループでの処理が行われるため，実環境での調査も必要となる．

次に，多様なネットワーク環境における実験の必要がある．本研究では有線，無線，無線と同じ帯域に制限した有線での実験しか行なっていないが，ジッタや帯域変動の激しいネットワークにおいてどのような挙動をするかがわからない．そのようなネットワーク環境をエミュレーションするなどして実際に提案手法がそのような環境でも有効であるかを調べる必要がある．

参考文献

[1] http://www.youtube.com/.

[2] ゼタバイト時代：トレンドと分析. http://www.cisco.com/web/JP/solution/isp/ipngn/literature/pdf/VNI_Hyperconnectivity_WP.pdf, May 2013.

[3] http://www.sec.gov/Archives/edgar/data/1418091/000119312513390321/d564001ds1.htm, October 2013.

[4] ニコニコ生放送. live.nicovideo.jp/.

[5] Ustream. http://www.ustream.tv/.

[6] 放送サービスの高度化に関する検討会. http://www.soumu.go.jp/main_content/000236951.pdf, May 2013.

[7] Iij technical week 2013 甲子園ライブ配信. http://www.iij.ad.jp/company/development/tech/techweek/pdf/131121_5.pdf, November 2013.

[8] A. Rao H. Schulzrinne and R. Lanphier. Real time streaming protocol (rtsp). *Network Working Group RFC 2326*, April 1988. http://tools.ietf.org/rfc/rfc2326.txt.

[9] Http live streaming の概要. https://developer.apple.com/jp/devcenter/ios/library/documentation/StreamingMediaGuide.pdf, August 2013.

[10] Ed. R. Pantos and W. May. Http live streaming draft-pantos-http-live-streaming-12. October 2013. http://tools.ietf.org/html/draft-pantos-http-live-streaming-12.

[11] Dash industry forum. http://dashif.org/mpeg-dash/.

[12] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE Multimedia*, Vol. 18, No. 4, pp. 62–67, April 2011.

[13] Thomas Stockhammer. Dynamic adaptive streaming over http –: Standards and design principles. *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, Vol. MMSys '11, pp. 133–144, 2011.

[14] Sola sphere. http://www.akamai.com/html/solutions/sola-sphere.html.

[15] Zixuan Zou Wei Pu and Chang Wen Chen. Dynamic adaptive streaming over http from multiple content distribution servers. *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–5, December 2011.

[16] ISO/IEC. Iso/iec 23009-1:2012 information technology – dynamic adaptive streaming over http (dash) – part 1: Media presentation description and segment formats. 2012.

[17] Yasuaki Kanatsugu Kinji Matsumura and Hisakazu Katoh. Toward the construction of hybridcast. *ATSC 2010 Symposium on Next Generation Broadcast Television*, September 2010.

[18] 金次保明松村欣司. Hybridcasttm の概要と技術. NHK 技研 R&D No.124, September 2010.

[19] Twivo. https://chrome.google.com/webstore/detail/twivo/ldpoeohapfjhfgiojngmleppigomkmfb/details.

[20] 1588-2008 - ieee standard for a precision clock synchronization protocol for networked measurement and control systems.

[21] 802.1as-2011 - ieee standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks.

[22] Cross-origin resource sharing. http://www.w3.org/TR/cors/, January 2014.

[23] 新世代通信網テストベッド jgn-x. http://www.jgn.nict.go.jp/.

[24] 【バルス祭り 2013】twitter/ニコニコ実況/2ch (3 窓)【天空の城ラピュタ】(2013.08.02) . http://www.youtube.com/watch?v=nZLPQJBSDrU.

[25] https://twitter.com/TwitterJP/status/363494742518013952.

[26] 栗原一貴. Cinemagazer: 動画の極限的な高速鑑賞のためのシステムの開発と評価. コンピュータソフトウェア, Vol. 29, pp. 293–304, 2012.

謝辞

本論文を執筆するにあたり，大変多くの方からご指導，ご協力をいただきました．ここに心より感謝の意を示します．

まず，いつも学生の見方になってご指導してくださった東京大学報理工学系研究科教授　江崎浩博士に感謝いたします．また，博士でありながらいつも学生に親身になってご指導・ご指摘をしてくださった浅井大史博士に感謝いたします．修士論文執筆の際にしていただいたご指摘によって大変助かりました．

研究に関する議論や質問にお付き合いいただきました土本康生博士，山本成一博士，落合秀也博士，土井祐介博士，塚田学博士，金海好彦博士に感謝いたします．

江崎研究室の OB として公私共にお世話になった白井俊宏氏，肥村洋輔氏，呉和賢氏，川口紘典氏，本舘拓也氏，Luciano Aparicio 氏に御礼申し上げます．

江崎研究室の先輩として苦楽をともにした正原竜太さん，林東權さん，小坂良太さん，石橋尚武さん，李聖年さんに感謝いたします．身近にいてわからないことを質問できるのは大変心強かったです．ありがとうございました．

また，江崎研究室の同期として研究室の仕事や作業をともに行ったの池上洋行君，川守田光昭君，美嶋勇太朗君，西田綾佑君，中村遼君，高成源君に感謝いたします．特に池上君とは議論することも多く力になりました．

江崎研究室のメンバーとして松田貴成氏，Buranachokphaisan Manussanun さん，大筒裕之くん，沼田進くん，小林諭くん，田中晋太朗くん，福田鉄平くんに感謝いたします．沼田くんには何かとものを壊されることが多かったですがそれもまた良い思い出です．Tran Quoc Hoan くん，東角比呂志くん，局成矢くん，中神啓貴くんに感謝いたします．留学生の Obi Ifeanyi Herbert さん，Julien Ribon さんに感謝いたします．

研究室生活を支援頂いた江崎研究室秘書の高橋富美さん，岩井愛映子さんに感謝致します．

正妻空母加賀さん，軽巡龍田に感謝いたします．

最後に，6 年間の大学生活を支えてくださった家族や友人，お世話になった皆様に深く感謝いたします．