TCP 並列接続を用いたプログレッシブダウンロードにおける順序制御方式の実装

広島市立大学 情報科学部 情報工学科

1420180　平城 光雄

概要

概要

ページ数	提出日	指導教員受付印	指導教員名
			舟阪 淳一

Implementation of sequence control method in progressive download using TCP parallel connection

Department of Computer and Network Engineering

Faculty of Information Sciences
Hiroshima City University

1420180 Mitsuo Heijo

Abstract
gaiyo

TCP 並列接続を用いたプログレッシブダウンロードにおける順序制御方式の実装

目次

第1章　はじめに

はじめに

第2章　関連研究

本章では関連研究について述べる.

2.1　TCP 接続を複数用いる HTTP

複数の TCP 接続を利用することで, より短時間でアプリケーションプロトコルとして mHTTP[4] が提案されている. mHTTP では DNS レコードを複数使用することで複数の TCP 接続を確立した上で, 既存の HTTP と同等の機能を提供する方式を提案している. この方式ではアプリケーション開発者やコンテンツプロバイダの負担を抑えながらも, 複数 TCP 接続を利用して効率的なダウンロードを実現している.

2.2　プログレッシブダウンロード方式

ネットワークの大容量化, 高速化に伴い,Youtube[5] や Netflix[6] などの動画配信サービスの利用が増加している. 動画配信サービスには UDP を利用したストリーミング,TCP を利用したプログレッシブダウンロードの 2 種類がある. アプリケーションプロトコルとして HTTP を用いるプログレッシブダウンロードは特別なソフトウェアを必要とせず, ブラウザだけで視聴することができるため, 近年広く普及してきている. また, プログレッシブダウンロードは分割順次ダウンロードとも呼称される. プログレッシブダウンロードの動作概要は, まず,1 つのファイルを複数のあるサイズのブロックに分割する. 次に, クライアントは分割されたブロックをサーバーに対してリクエストする. サーバーはリクエストに応じたブロックを送信する. これを繰り返すことで,1 つのファイルを取得できる. このリクエストの方法には HTTP の Range-Header に分割のための情報を含める方式や HTTP の GET リクエストのクエリストリングに分割のための情報を含める方式などがある. 前者は Range-Header に対応した HTTP サーバーソフトウェアがあれば特別な設定をすることなく利用が可能である. 後者はフロントエンドおよびバックエンドでの実装が必要であるが, 状況に応じて動画の品質を変化させたりするなどしたサービスに応じた柔軟なコントロールが可能である.Youtube ではこちらが用いられている.

2.3　複数のTCP接続を用いたプログレッシブダウンロード

ネットワークの発展に伴い, 大容量のデータを TCP を用いて, 通信する機会が増加しつつある.TCP には輻輳回避のためにウィンドウ制御が存在する. このため, ウィンドウサイズを遅延で割ったものが単一 TCP 接続における理論最大性能となる. 近年ではコンテンツの大容量化が進んでおり, より効率よくコンテンツをダウンロードするためにアプリケーション層から複数の TCP 接続を用いる手法が提案されている. 図 2.1 は複数の TCP を接続を用いたプログレッシブダウンロードの分割されたブロックの受信の様子を示した例である. 複数のブロックを性能の異なる別々の TCP 接続に対して要求を行う場合, ブロックの再生順番と受信完了順序が一致しない可能性がある. 図 2.1 が示すように, 先頭から連続するブロック 1 及びブロック 2 は再生可能である (有効ブロック) が, それ以外のブロック 3 及びブロック 5 は未受信のブロックを間に挟んでいるため再生することはできない (非有効ブロック). 複数の TCP 接続を束ねることでグッドプットを向上させても, 受信ブロックが有効ブロックでない限りは応答性は低下してしまい, 動画の再生が停止するなどしてユーザー体験は悪化することが予想できる.

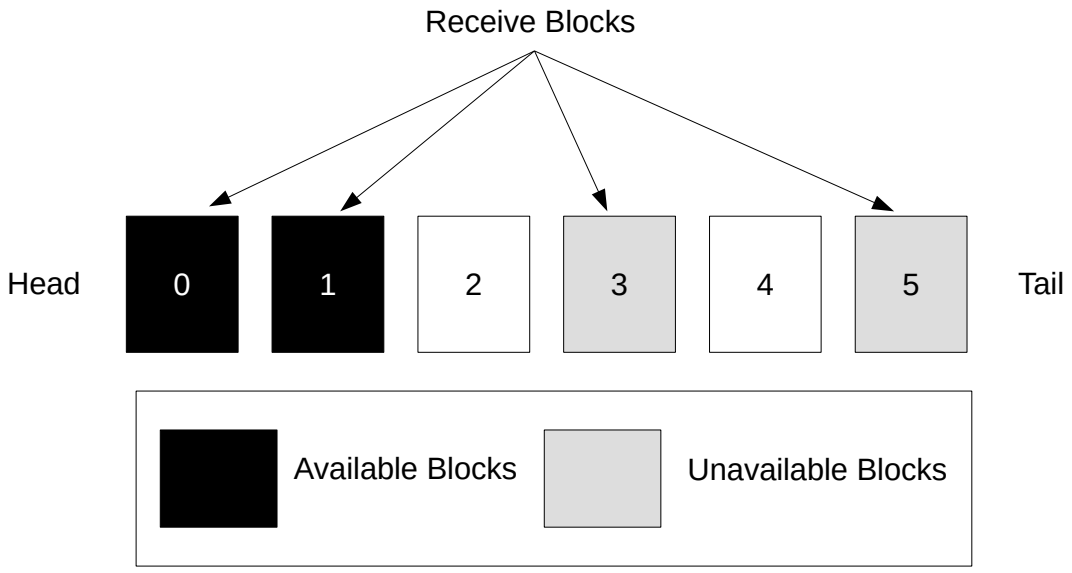

図 2.1: ブロックの有効性

2.4　重複再要求

2.3 節で述べた性能差のある複数の TCP 接続を用いたプログレッシブダウンロードにおいて起こりうる問題点を, アプリケーション層での制御で解消するために提案されている方式として, 重複再要求 [] がある. この方式では未取得ブロックより後に合計 N 個以上 (有効・非有効は問わない) のブロックがあれば, 未取得ブロックをその未取得ブロックを要求した TCP 接続とは別の TCP 接続へ再要求を行う. 図 2.2 にその模式図を示す. 図 2.2 の例では重複再要求を行い, ブロック 2 を取得することで少なくともでもブロック 3 が非有効ブロックである状態を解消することができる. この操作を受信イベントが発生するたびに繰り返すことで, バッファ上の非有効ブロックの個数の増加を抑制することができ, 応答性の向上が見込まれる.

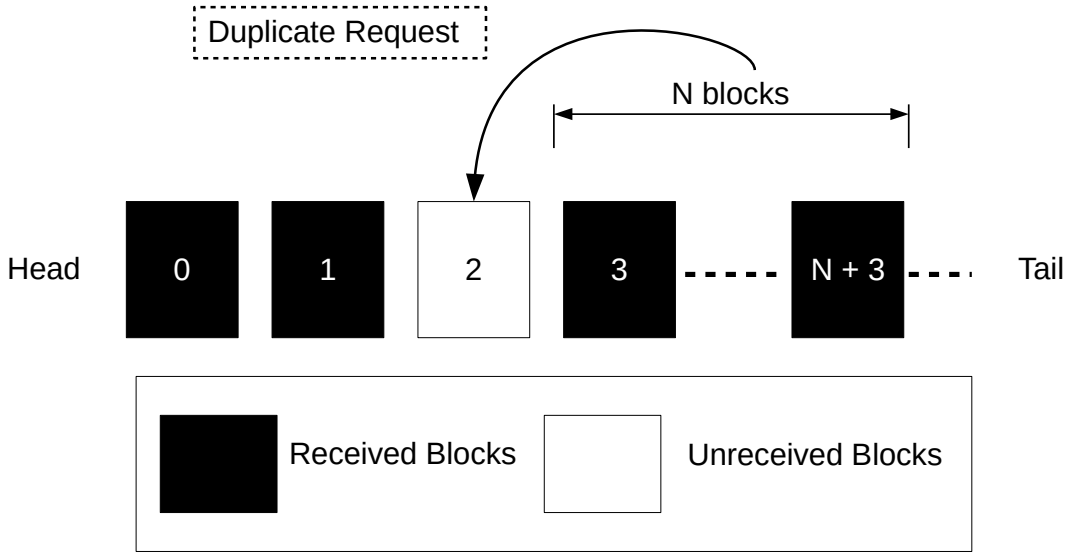

図 2.2: 重複再要求の模式図

2.5　タイマ駆動を用いた要求方式

2.3 節で述べた複数の TCP 接続を用いたプログレッシブダウンロードにおいて起こりうる問題点を解消するために提案されている方式として, タイマ駆動型要求方式 [] がある. 2.4 節で述べた重複再要求方式は, ブロックの遅延に対して後から対処するという方針であるが, 本節で述べるタイマ駆動を用いた要求方式は,TCP 接続の性能差を予め考慮することで, 到着順序逆転の発生そのものを抑制しようという方針である. はじめに比較対象となる受信駆動を用いた要求方式について説明する. 受信駆動とは, あるTCP 接続に対して常時 1 つのブロックを要求する方式である。ブロックの受信が終了したら次ブロックに対する要求を送信する. タイマ駆動とは前ブロックの要求送信からある時間が経過したら次ブロックへの要求を行う. このとき前ブロックの受信完了を待つことはない. 以下に模式図を示す. 先行研究としてはタイマ駆動を用いた要求方式において,TCP 接続間の性能差を要求の送信間隔に反映させる方式が提案されている.

第3章　提案方式

本章では性能差のある複数の TCP 接続を並列的に利用する際に生じうるいくつかの問題点を解消するためのアルゴリズムを実装した提案方式について述べる. また, 実際に複数の TCP 接続を用いた動画のプログレッシブダウンロードをプログラムに実装する際に考慮すべき点がいくつかある. プログレッシブダウンロードの実装は大きく分けて動画ファイルのダウンロードとバイナリファイルをデコードして再生という 2 つのセクションに分かれている. 既存のウェブブラウザや VLC[] 等のネットワークメディア再生機能付きの動画プレイヤーソフトのではこの 2 つのセクションは 1 つのプログラムから高度に同期をとりながら同時並列的に制御されている. しかし, 本研究では実装の難易度と主としてダウンロードセクションについて論じるためにこれら 2 つのセクションは分離している.

3.1　遅延要求方式

3.1.1 節では遅延要求の概要について述べる. 3.1.2 節では各接続の帯域が既知であるいう仮定に基づいて,TCP 接続の性能差を入力し, ブロックの要求位置を変化させることで到着順序逆転の抑制する方式について提案する. 3.1.3 節では未知のネットワーク状況に対応するために TCP 接続の使用回数の差分に注目しブロックの遅延度を推測する方式について提案する.

3.1.1　遅延要求について

この章で定義する遅延要求についての概要を述べる. まず, 確立した TCP 接続群の中で性能の最も高い TCP 接続には, 最も若番のブロックを要求する. 続いて, 比較性能の低いある TCP 接続に関して, ブロック要求の送信からブロックの到着までの間隔を算出し, その算出値に基づいてその時点での最も若番のブロックではない後ろのブロックを要求する. 図 3.1 にその模式図を示す. この例では, 接続 A は接続 B の 2 倍の性能を持つと仮定する. この条件より接続 B がブロックを 1 個取得する間に接続 A はブロッ

クを 2 個取得取得することが予想できる. よって時刻 t=0 に接続 A にはブロック 0 を要求し, 接続 B にはブロック 2 を要求する. t=1 には接続 A にブロック番号 0 が到着し, 続いて接続 A がにブロック 1 を要求する. 時刻 t=2 において接続 A には 2 より若い番号のブロックが到着済みであり, 接続 B はブロック 2 の取得が完了する. よってブロックの到着順序逆転を抑制することができる. 3.1.2 節ではあらかじめ各接続間の性能差が既知であるとして, その値にしたがって遅延要求を行う方式について述べる. 3.1.3 節では各接続ごとにブロック到着間隔を逐次測定することで性能差を算出し, その値にしたがって遅延要求を行う方式について述べる.

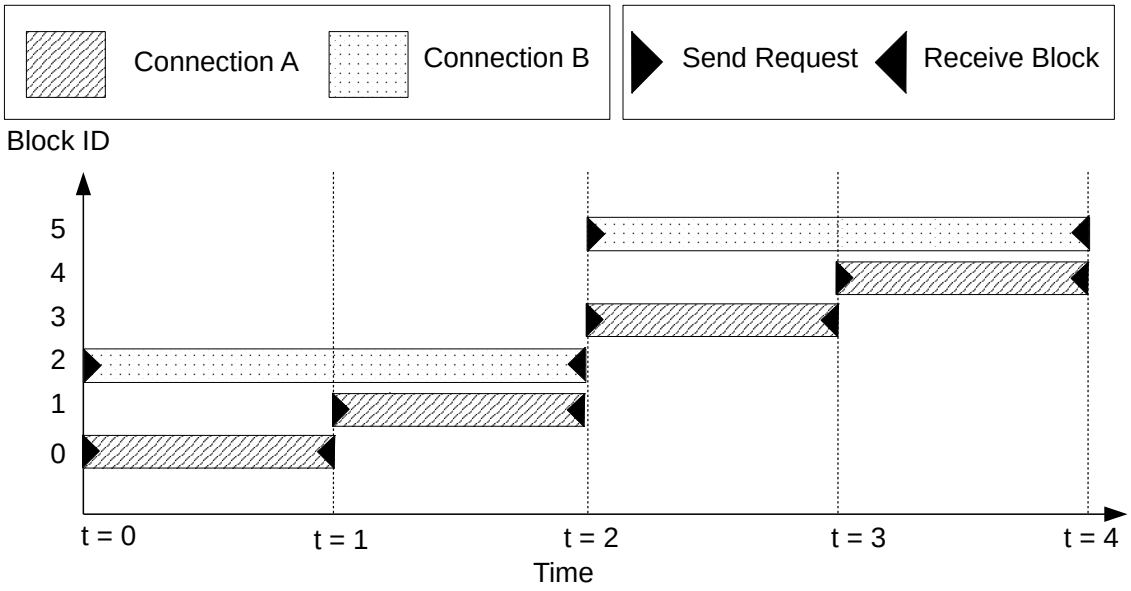

図 3.1: 遅延要求の模式図

3.1.2　固定遅延要求方式

固定遅延方式は, 予め TCP 接続間の性能差をシステムに入力して遅延要求を行うことで到着順序逆転の抑制を試みる方式である. しかし, 実ネットワークでは予め TCP 接続間の性能差が既知であることは稀であるので実環境への応用は限定的であると言わざるを得ない. よって本研究では本方式は主として, 他の方式がどれだけ TCP 接続間の性能差を正確に把握できているかどうかを比較し確認するために用いる.

3.1.3　差分計測を用いた遅延予測方式

当方式はある TCP 接続に対して何ブロック後ろのブロックを要求するかを算出するために, その TCP 接続の直前のブロック取得間隔を計測し用いる方式である. 以下に疑似コードを示す. 最も性能の高い TCP 接続には 0 を割り当てる.

Algorithm 1 Compute Diff

```
T ← Total Receive Count
P ← Previous Receive Counts
N ← Number Of Connections
if Is it the highest performance then
    D ← 0
else
    D ← T − P − N
end if
```

3.2　初期遅延予測

3.1.3 節の遅延要求方式は実装上の都合, 最初のリクエスト送信の際には TCP 接続間の性能差が不明であるため, 遅延要求を行うことができない. しかし実際にエンドユーザーが動画再生を行うことを想定すると, 初期バッファリング時間の長さは視聴体験に大きく影響を及ぼすことが予想される. 本節ではこの問題の解決案として初期値を予測し初期バッファリング時間の短縮を目指す方式を提案する.

3.2.1　概要

3.1 節の遅延要求方式ではファイル情報を取得するために, 事前に HTTP の HEAD リクエストを相手サーバーに送信している. この HEAD リクエストの応答時間を計測することで, 各 TCP 接続間の性能差も推測できる. ただし, 後続の GET リクエストの応答メッセージサイズは HEAD レスポンスのそれよりも大きく, 実際には一つの HTTP レスポンスに対して複数の TCP セグメントがやり取りされるので, 厳密な意味での応答時間ではなく単一の HEAD レスポンスの応答時間であることに注意が必要である.

3.2.2　アルゴリズム

本節では具体的な初期遅延予測アルゴリズムについて述べる.

初期遅延予測係数

ここで定義する遅延予測係数とは, 各 TCP 接続での HTTP HEAD リクエストの応答時間の比を TCP 接続の帯域性能差へ変換するための係数である. TCP のウィンドウサイズがネットワーク状況に応じて変化することや高遅延でありながらも広帯域のネットワークが実際に存在することを考慮すると, 初期応答時間から TCP 接続の性能を単純に予測することは現実的ではない. しかし, 仮に広帯域であっても高遅延なネットワークでは TCP のウィンドウサイズが大きくなるまでに低遅延ネットワークよりも長い時間がかかる. また, 初期ブロックの到着時刻がユーザー体験に与える影響は大きいと考えられる. よって初期リクエストの送信時に限って言えば, 多少の性能予測が外れることは許容しても, 低性能の可能性がある高遅延な TCP 接続にはとにかく初期ブロックを要求させないことが, 初期バッファリング時間の短縮につながる. 各接続における初期リクエスト-レスポンスさえ終了してしまえば, そこからは性能計測は差分予測の役割になる. つまり, 応答遅延時間から最悪ケースとして冗長的に TCP 接続の性能を見積もることで, ユーザー体験の悪化を防ぐことが初期遅延予測係数の役割である.

算出

初期遅延度は各 TCP 接続において計測した HEAD リクエストの応答時間と, すべての TCP 接続の応答時間の最小値との比に初期遅延予測係数を掛けることで算出する. 以下に疑似コードを示す.

Algorithm 2 Compute Initial Delays

```
1: R ← Raw Delays
2: M ← MIN(R)
3: D ← Delays
4: C ← Coefficient
5: for all r in R do
6:     d ← (r / M − 1) * C
7:     Add d to D
8: end for
```

第4章　実装評価

本章では提案方式を実装し, 評価する.

4.1　評価対象

本評価では重複再要求方式として 2.4 を改良し非有効ブロックの受信回数を閾値として用いる. これは実装上の都合であり, 関連研究のアルゴリズムとは若干異なるが本研究では重複再要求のアルゴリズムを比較することは目的としていない. すでに重複再要求方式の有効性は先行研究において確認されている. よって以降全ての実験は重複再要求を有効にして行うこととする.

4.2　評価項目

本章で評価する評価値について整理する.

表 4.1: 評価項目

評価項目	概要
初期バッファリング時間	初期バッファリングに必要な時間
平均非有効ブロック数	バッファ内の非有効ブロックの数

4.3　テストベッドでの評価

4.3 にネットワーク環境を示す.TCP 接続 A-TCP 接続 B 間の性能差は 3 倍,TCP 接続 A-TCP 接続 C 間の性能差は 10 倍である.

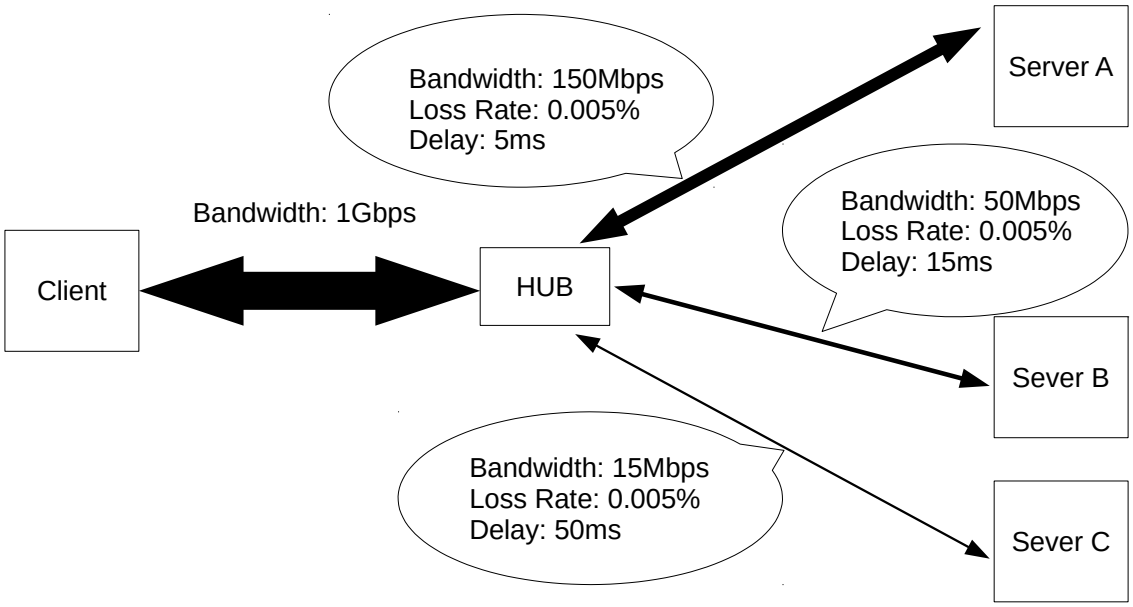

図 4.1: ネットワーク環境

実験環境および実験パラメータを表 4.2, 表 4.3 に示す.

表 4.2: 実験環境

ファイルサイズ	754MByte
ファイル	ubuntu-17.10.1-server-amd64.iso
OS(Server and Client)	ubuntu 17.10 (Kernel 4.13)
TCP	CUBIC
HTTP Server	h2o v2.2.4

表 4.3: 実験パラメータ

ブロックサイズ	10^6 Byte
初期遅延予測係数	10
重複再要求発行閾値	20

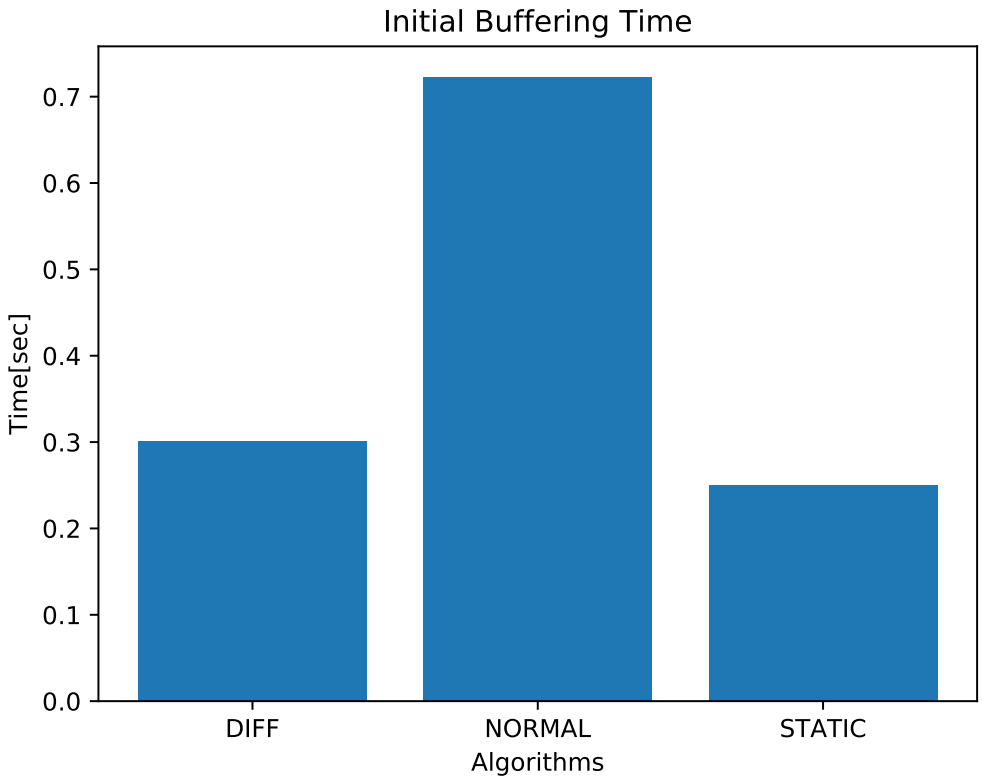

図 4.2: 初期バッファリング時間

図 4.2 は遅延要求のアルゴリズムごとの初期バッファリング時間である. DIFF は差分計測を用いた遅延要求方式,NORMAL は常時最若番を要求する方式 (制御を何も行わない),STATIC は TCP 接続間の性能差を入力する固定遅延要求方式である. 図 4.2 より差分計測を用いた遅延要求方式は TCP 接続間の性能差を入力することなく固定遅延要求方式に近い性能を獲得できていると言える.

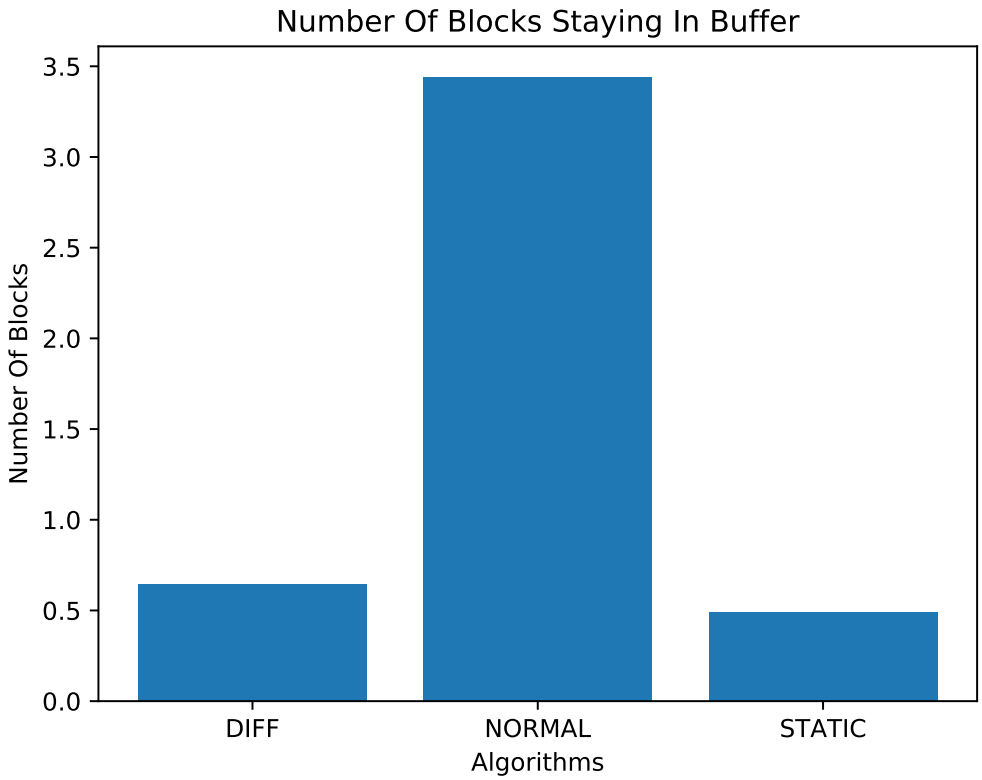

図 4.3: 平均非有効ブロック数

図 4.3 は遅延要求のアルゴリズムごとの非有効ブロック数である．DIFF は差分計測を用いた遅延要求方式,NORMAL は常時最若番を要求する方式 (制御を何も行わない),STATIC は TCP 接続間の性能差を入力する固定遅延要求方式である．図 4.3 より差分計測を用いた遅延要求方式は TCP 接続間の性能差を入力することなく固定遅延要求方式に近い性能を獲得できていると言える．

4.4　パブリックネットワークでの評価

パブリックネットワークでの評価にあたり,Ubuntu のリリースイメージファイルの配布に用いられているパブリックミラーを利用した. 表 4.4 に使用したパブリックミラーを示す. また, 参考として各パブリックミラーの 24 時間の性能変化について図 4.4 に示す. 実験の際にダウンロードしたファイルおよび実験パラメータはテストベッドのものと同一である. 図 4.4 からわかるように 15 時から 18 時においてネットワークの変化が大きいと判断しこの時間帯で実験を行った.

表 4.4: 使用したパブリックミラー一覧

ホスト	組織	国
ftp.jaist.ac.jp	JAIST	JP
ubuntutym2.u-toyama.ac.jp	Univercity of Toyama	JP
releases.ubuntu.com	Canonical	GB
mirrorservice.org	University of Kent	GB
ubuntu.ipacct.com	IPACCT	BG
mirror.pop-sc.rnp.br	PoP-SC	BR
ftp.belnet.be	Belnet	BE
mirrors.mit.edu	MIT	US
mirror.yandex.ru	Yandex	RU

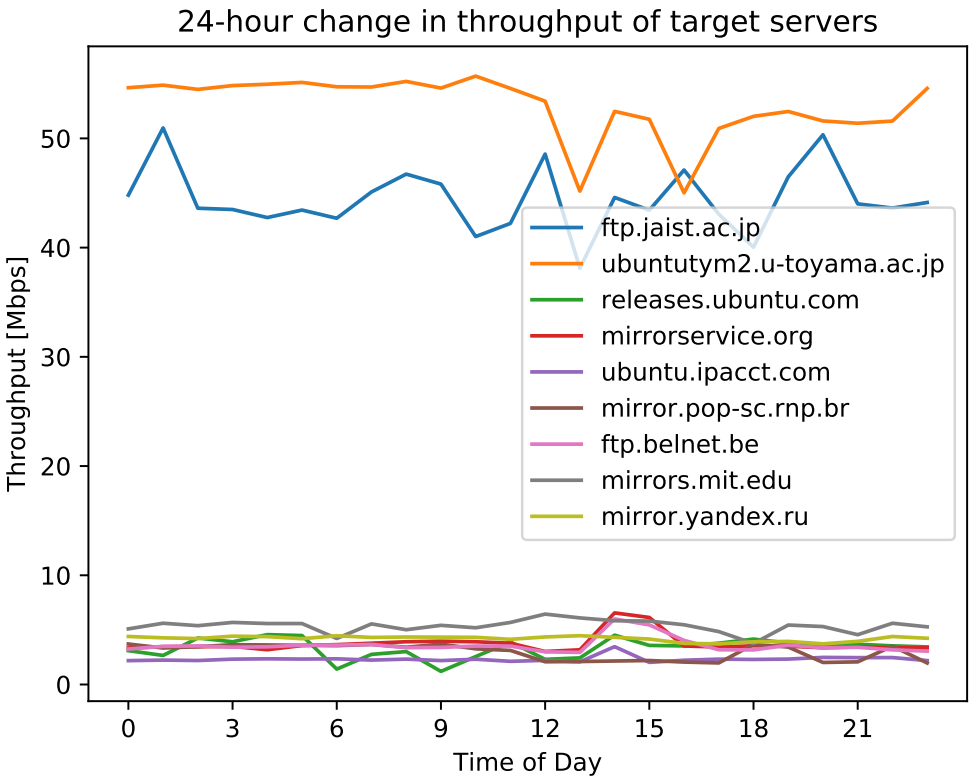

図 4.4: 各パブリックミラーの 24 時間の性能の変化

典型的な受信の様子についてのグラフを図 4.5 および図 4.6 に示す.

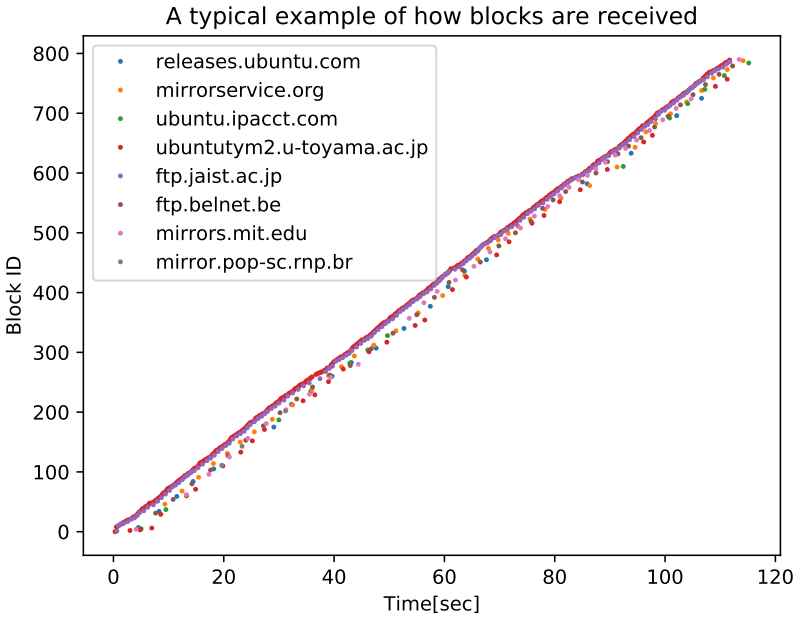

図 4.5: 遅延要求なしの場合の受信の様子

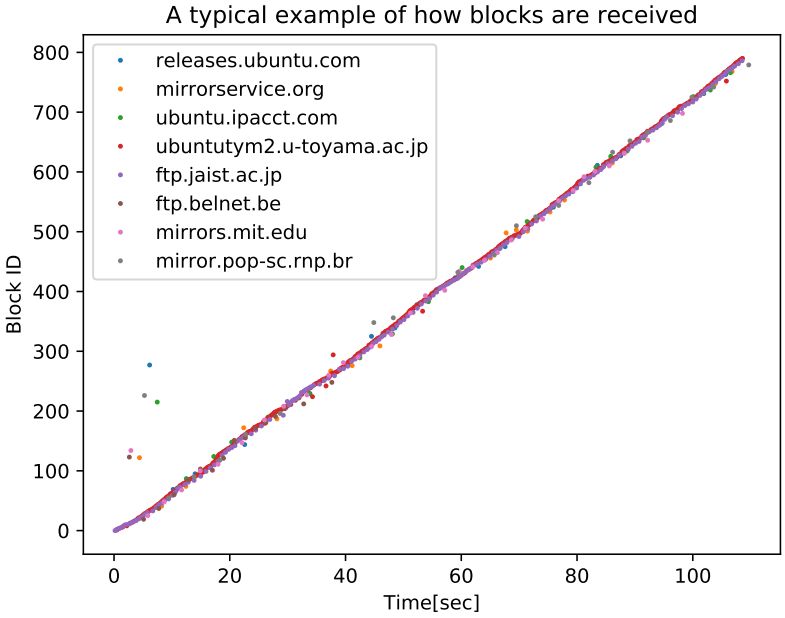

図 4.6: 差分計測を用いた遅延要求をおこなった場合の受信の様子

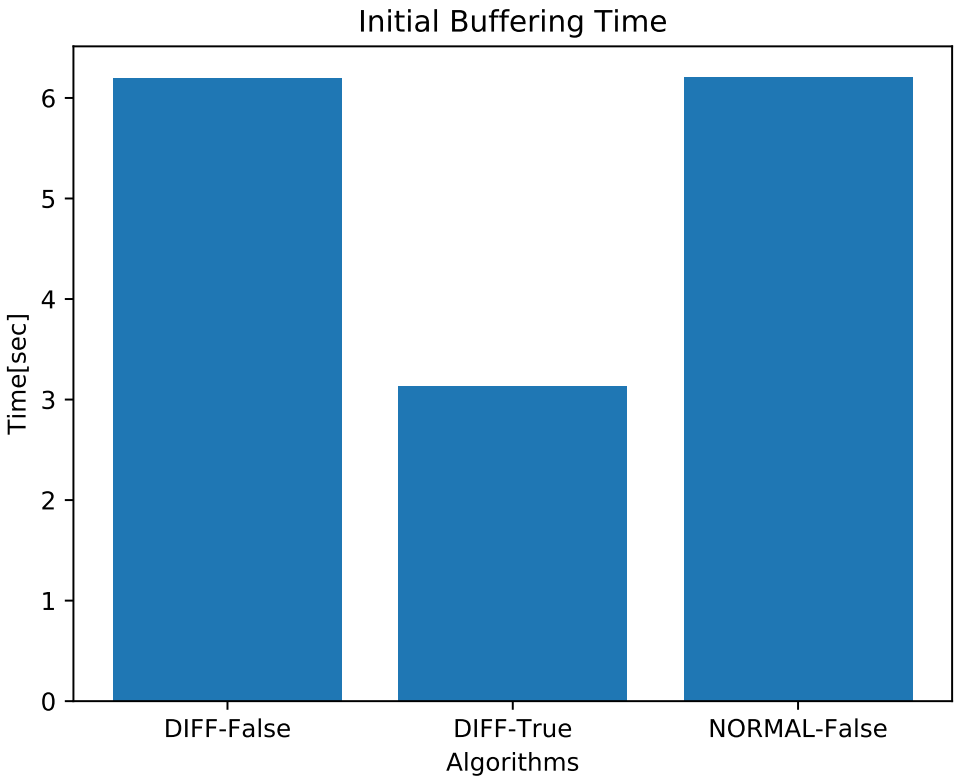

図 4.7: 初期バッファリング時間

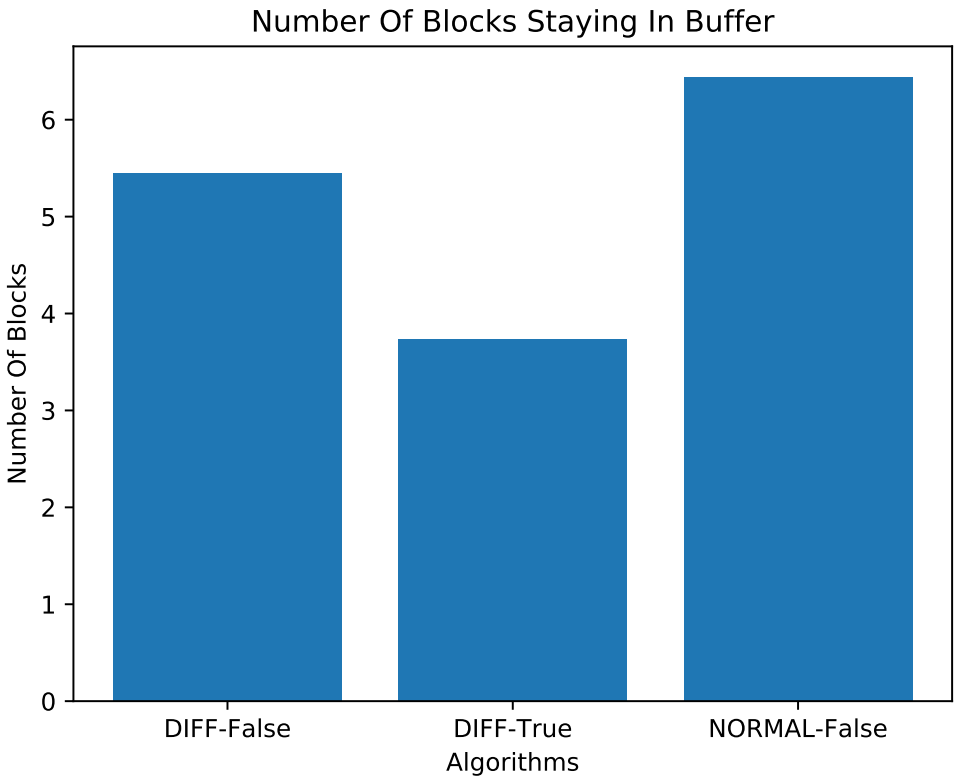

図 4.8: 平均非有効ブロック数

第5章　まとめと今後の課題

本章では, 本論文のまとめおよび今後の課題について示す.

5.1　まとめ

本研究では TCP 接続を並列に用いたプログレッシブダウンロードにおいて TCP 接続間に性能差が存在する場合に生じる問題点を解消する方式を提案し実装を行った. 提案方式は, 各 TCP 接続におけるブロックの到着間隔を計測することでその TCP 接続に遅延要求を行うことで到着順序の逆転を抑制する方式である. 次に実装したプログラムをテストベッドおよびネットワーク環境が未知の公開ネットワーク上で動作させて評価を行った. テストベッドでの評価では, 差分要求方式は TCP 接続間の性能差を入力することなく固定遅延要求方式と同等の性能を獲得することができた. 続いてパブリックネットワークでの評価では差分計測を用いた遅延予測方式は初期遅延予測方式と組み合わせることで制御なしの場合と比べて,50%の初期バッファリング時間の削減と 30%の非有効ブロック数の削減が確認できた.

5.2　今後の課題

今後の課題として, 以下が挙げられる.

- 実際のユーザー体験を考慮した評価
- タイマ駆動要求方式の実装との比較

謝辞

本研究の機会を与えて頂き, 多くの御指導, および御助言を賜わりました舟阪 淳一准教授に深甚なる謝意を表します. また, その他多くの御助言を頂きました諸氏に心より感謝致します.

参考文献

[1] Junichi Funasaka, Atsushi Kawano, and Kenji Ishida: Adaptive Parallel Downloading Method for Proxy Systems, IEICE Trans., Vol.E90-B, No.4, pp.720-727, Apr. 2007.

[2] Tokumasa Hiraoka and Junichi Funasaka: A Progressive Download Method Using Multiple TCP Flows on Multiple Paths, Proc. 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA 2015), pp.318-324, Nov. 2015.

[3] Hiroaki Horiba, Tokumasa Hiraoka, and Junichi Funasaka: A Progressive Download Method Based on Timer-Driven Requesting Schemes Using Multiple TCP Flows on Multiple Paths, Proc. 37th IEEE ICDCS Workshops, pp.26-31, Jun. 2017.

[4] Juhoon Kim, Yung-Chih Chen, Ramin Khalili, Don Towsley, Anja Feldmann, "Multi-source Multipath HTTP (mHTTP): A Proposal," ACM SIGMETRICS Performance Evaluation Review, vol. 42, No. 1, pp.583–584, 2014.

[5] Youtube.available at https://www.youtube.com,2018.

[6] Netflix.available at https://www.netflix.com,2018.

[7] Ubuntu Mirrors.available at https://launchpad.net/ubuntu/+cdmirrors, 2018.