平成 27 年度　修士論文

NDN 上のビデオ配信における
再生バッファ制御に基づく電力モデル提案と
省電力化への検討

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻

5114F010-0
石津　裕也

指導　甲藤 二郎　教授

2016 年 2 月 1 日

指導教授印	受付印

目次

第 1 章 序論 4
1.1. はじめに 4
第 2 章 ビデオストリーミング 6
2.1. ビデオストリーミングの基本的問題点([3]) 6
2.1.1. 帯域 6
2.1.2. ジッタ 7
2.1.3. パケットロス 7
2.2. ビデオストリーミングプロトコル 7
2.2.1. RTP(Real Time Protocol) 8
2.2.2. RTCP(Real Time Control Protocol) 8
2.2.3. RTSP(Real Time Streaming Protocol) 10
2.2.4. HTTP Live Streaming 11
2.2.4.1. サーバコンポーネント 12
2.2.4.2. ディストリビューションコンポーネント 13
2.2.4.3. クライアントコンポーネント 14
2.2.5. MPEG Dynamic and Adaptive Streaming over HTTP 14
2.2.5.1. MPD(The Media Presentation Description) 15
2.2.6. DASHEncoder 17
2.2.7. DASH-JS 18
第 3 章 NDN(Named Data Networking) 21
3.1. NDN における通信手法の概要 21
3.2. NDN ノードの構成 22
3.2.1. PIT(Pending Interest Table) 22
3.2.2. FIB(Forwarding Information Base) 22
3.2.3. CS(Content Store) 22
3.3. NDN オープンソースアプリケーション 24
3.3.1. NDNx/CCNx 24
3.3.2. NDN-TLV 24
3.3.3. NDN-JS 24
3.3.3.1. NDN-JS を利用したコンテンツダウンローディング 24
第 4 章 TCP の輻輳制御方式 28
4.1. TCP Tahoe 28

4.2. TCP Reno 29
4.3. TCP Vegas 30
4.4. CUBIC TCP 31
第 5 章 NDN における輻輳制御方式性能評価 33
5.1. NDNx の輻輳制御方式 33
5.1.1. NDNx の RTT 推測機能 34
5.1.2. NDNx の RTT について 34
5.2. 提案方式① 35
5.2.1. RTT-Base 方式 35
5.2.2. TCP Vegas 方式 35
5.2.3. FAST TCP 方式 35
5.3. 評価実験 35
5.3.1. 実験環境 36
5.3.2. 実験結果 37
第 6 章 Interest アグリゲーションと再生バッファを利用した NDN における省電力ビデオストリーミング 40
6.1. ビデオストリーミングにおける再生バッファ制御 40
6.2. Wi-Fi 接続時の端末消費電力 41
6.3. NDN における効率的動画配信 42
6.3.1. Method 42
6.3.2. 実装と実験環境 43
6.3.3. 実験結果 44
6.4. 再生バッファ制御による省電力ビデオストリーミングの検討 46
6.4.1. Method 46
6.4.2. 実験環境 47
6.4.3. 実験結果 48
第 7 章 ビデオ配信における再生バッファ制御に基づく電力モデル提案と省電力化への検討 49
7.1. スマートフォンの LTE 状態遷移 49
7.1.1. スマートフォンの LTE 状態遷移と端末消費電力 50
7.2. ビデオストリーミング時の端末消費電力のモデル化 51
7.2.1. 通信スループットと端末消費電力の関係 54
7.3. 消費電力量モデルの検証 57
7.3.1. 実験環境 57
7.3.2. 実験結果および考察 59

第 8 章 総括 61
8.1. まとめ 61
8.2. 今後の展望 61
参考文献 62
謝辞 64
発表文献リスト 65

第 1 章 序論

1.1. はじめに

近年、タブレット端末やスマートフォンのような携帯端末の普及・多様化が進み、その携帯端末の処理能力も向上している。また、モバイルビデオコンテンツは他のモバイルコンテンツに比べビットレートが高いため、2019 年までに約 72%を占めるということがシスコシステムズにより予測されている[1]。また、2014 年から 2019 年までの CAGR(Compound Average Growth Rate)は 66%と、急速な増加が推測されている。今後、4K、8K といったさらに解像度の高い動画像をユーザが視聴する機会の増加や、スマートフォンの処理機能の向上により、ユーザのビデオストリーミング時の端末消費電力削減は重要な要素となってくると考えられる。また、今後のビデオコンテンツのリッチ化に伴い、端末消費電力のうち、通信消費電力の占める割合が大きくなり、通信部分の消費電力削減がより重要なものとなると考えられる。

本稿では、コンテンツの効率的配信を実現するために近年提案された NDN(Named Data Networking)上でのビデオストリーミング時に加え、IP ネットワーク上でのビデオストリーミング時のクライアント端末消費電力量の削減を目的とした検討を行う。

1.1. 本論文の構成

第 2 章では、ビデオストリーミングについて述べる。
第 3 章では、NDN(Named Data Networking)について述べる。
第 4 章では、TCP の輻輳制御について述べる。
第 5 章では、NDN における輻輳制御方式の改善について述べる。
第 6 章では、Interest アグリゲーションと再生バッファ制御を利用した NDN のおける省電力ビデオストリーミングについて述べる。
第 7 章では、ビデオ配信における再生バッファ制御に基づく電力モデル提案と省電力化への検討について述べる。
第 8 章では、まとめと今後の展望について述べる。

第 2 章 ビデオストリーミング

従来は、ビデオデータという膨大なデータをインターネット上から端末へすべてダウンロードしてからでないと動画を再生することができなかった。動画という膨大なデータのダウンロードには時間がかかり、その動画を視聴するためにはユーザはダウンロードの終了まで待たなければならないという問題が存在した。その問題を解決する技術がストリーミング技術である。ビデオストリーミングの基本的な概念は、以下の 3 つのステップに分けられる。

1. 符号化されたビデオコンテンツをパケットに分割する
2. パケットを配信する
3. ストリーミングクライアントは、パケットを受信しつつ、受信したパケットからデコードと再生を開始する

以上のメカニズムにより、クライアントは動画データ全体のダウンロード待つことなく動画を再生することができる。この技術は、再生が始まるまでの遅延の削減や、少量のパケットのみをバッファにため込むため、使用するクライアントのストレージが小さいといったメリットが存在する。

2.1. ビデオストリーミングの基本的問題点([3])

ビデオストリーミングを実現するにあたり、様々な問題が存在する。インターネットは帯域、遅延ジッタ、パケットロス率などについて保証していないため、インターネット上でのビデオストリーミングの実現は困難とされてきた。また、特に以上 3 つの要素がインターネット上でのビデオストリーミングを困難なものにさせる原因として、不明であり、動的であることである。ビデオストリーミングの実現には、以上 3 つのパラメータを考慮し、信頼性のある高品質な動画配信アーキテクチャを設計する必要がある。

2.1.1.帯域

インターネット上の特定の 2 点間の利用可能帯域幅は一般的に不明であり､時変である。サーバが利用可能帯域幅以上の速度でデータを送ってしまうと、輻輳やパケットロスが起き、ビデオストリーミングの品質の劣化に繋がってしまう。また、サーバが利用可能帯域幅より低い速度でデータを送ってしまうと、ユーザは最適下限のビデオストリーミング品質となってしまう。以上の帯域に関する問題の解決目標は、利用可能帯域幅を正確に推測し、それに対して最適なビットレートのビデオコンテンツを配信することである。

2.1.2.ジッタ

エンド間の遅延はパケットによって異なる。このエンド間の遅延のばらつきのことを遅延ジッタと呼ぶ。クライアントは一定の速度で受信、デコード、ディスプレイを行わなければならないため、遅延ジッタにより遅れたフレームは動画のフリーズの原因等になってしまう可能性がある。この問題は、図 2.1 のように再生バッファをクライアント側に設けることにより解消することができる。図 2.1 より、遅延の極端に大きい場合などはパケットロスが起こっているが、大半の場合において再生まで時間を設けることによって遅延ジッタを許容していることがわかる。

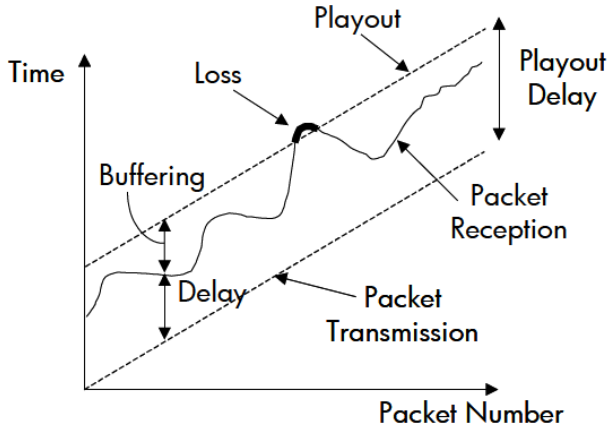


図 2.1 遅延パケット数に対する再生バッファの効果([3]より)

2.1.3.パケットロス

パケットは特定のネットワークにおいて様々な要因により廃棄されることがある。有線ネットワークでは、パケットロスが起こった場合、パケット全体が失われてしまう。一方、無線ネットワークにおいては、特にビットエラーまたはバーストエラーによってよってパケットが失われる。以上のようなパケットロスは動画再生にとって大きな障害となる。そこで、ビデオストリーミングはパケットロスを克服するため、FEC(Forward Error Correction)、再送、Error Concealment、Error-Resilient Video Coding といった、大きく分けて 4 つのエラーコントロールと共に設計されている。

2.2. ビデオストリーミングプロトコル

図 2.2 は IP ネットワーク上で RTP ストリーミングを実現するプロトコルスタックを示している。ビデオやオーディオといったメディアの再生・停止・早送り・巻き戻しなどを提供する RTSP(Real Time Streaming Protocol)は信頼性に優れた TCP 層の上に位置する。

一方、メディアデータそのものの転送を行う RTP や、その制御を行う RTCP はリアルタイム性に優れた UDP 層の上に位置している。

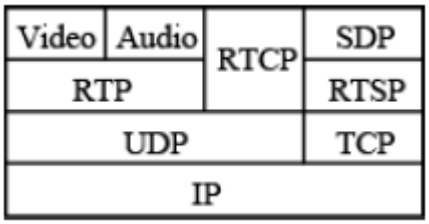


図 2.2 RTP、RTCP のプロトコルスタック([4]より)

2.2.1.RTP(Real Time Protocol)

UDP は信頼性がないプロトコルであり、パケットの損失や、順番の入れ替わりが起こる可能性がある。UDP でリアルタイムなマルチメディア通信を実現するためには、パケットの順番を表すシーケンス番号を付け、パケットの送信時刻を管理する必要がある。RTP はそれぞれのパケットにタイムスタンプとシーケンス番号を付加する。パケットを受け取ったアプリケーションは、このタイムスタンプの時刻を基に再生するタイミングを調整することができる。シーケンス番号はパケットを送るごとに 1 つずつ増やされる。シーケンス番号を使って同じタイムスタンプを持つデータの順番を並べ直すことを可能にし、パケットの抜けを把握することができる。図 2.3 に RTP ヘッダの構造を示す。



図 2.3 RTP ヘッダの構成([2]より)

2.2.2.RTCP(Real Time Control Protocol)

RTCP とは RTP と組み合わせて使う、データのフロー(送受信)制御および送信者と受信者の情報を記述するための仕様である。RTP は、音声や動画などの動画ストリームをリアルタイムに配送するためのデータ転送プロトコルであり、RTCP は RTP データを送受信するためのセッションを制御するプロトコルである。データストリームの受信者が RTCP パケットを定期的に送信することで、送信者の伝送レートなどの調整を行う。RTCP は IP マ

ルチキャストを用いた音声や動画通信を行う様々なアプリケーションに実装されている。図 2.4 に RTCP ヘッダの構造、図 2.5 に送信レポートパケットの構造、図 2.6 に受信レポートパケットの構造を示す。

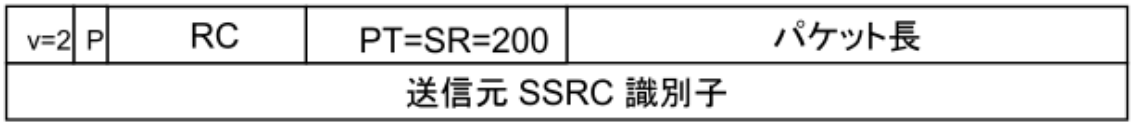


図 2.4 RTCP ヘッダの構造([2]より)

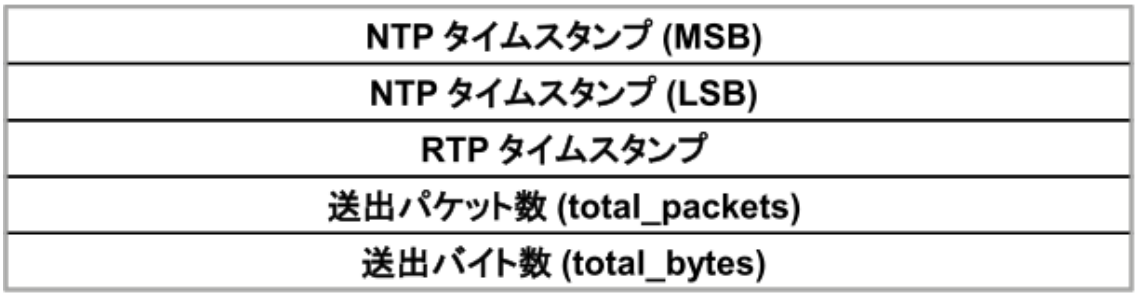


図 2.5 送信レポートパケットの構造([2]より)



図 2.6 受信レポートパケットの構造([2]より)

図 2.5, 図 2.6 より、送信レポートパケットはストリームに関する情報を保持し、受信レポートパケットは受信者での受信品質に関する情報を伝えるために利用される。これらを利用して送信側は受信側の受信品質を知ることができる。RTP・RTCP の使い方を図 2.7 に示す。

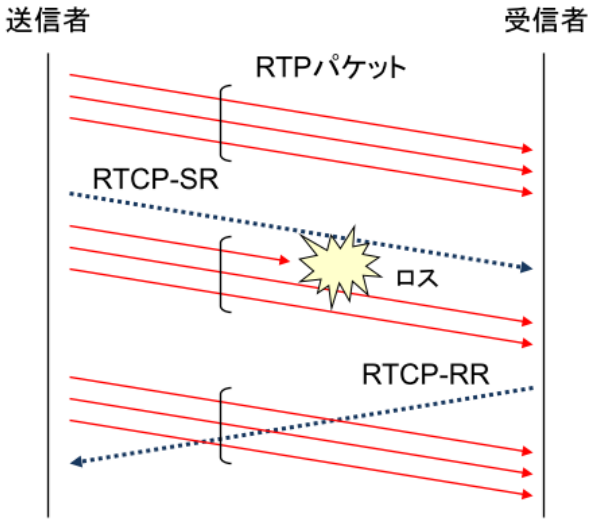


図 2.7 RTP・RTCP の使い方([2]より)

図 2.7 のように RTCP パケット(RTCP-SR、RTCP-RR)を使用することで送信側は受信側の受信品質を知ることを実現している。具体的にはパケットロスと RTT がわかる。受信側でのパケットロスは RTCP-RR に記述され、RTT は RTCP-SR と RTCP-RR の両方を組み合わせて得られる。送信者は RTCP-SR を送信した時間を覚えておく。受信側は、RTCP-SR を受け取ってから RTCP-RR を送信するまでの時間を計測し、その値を RTCP-RR に記録する。送信者は、RTCP-SR を送った時間と RTCP-RR を受け取った時間の差を計算し、それが RTT に相当する。

また、RTCP はメディア間同期機能を持つ。動画や音声などのメディア内では RTP スタンプを使用して同期を行うが、メディア間の再生がずれてしまう可能性がある。これを改善するのが図 2.5 に示されている NTP タイムスタンプであり、この NTP(共通)時間軸を使用してメディア間同期が行われる。

2.2.3.RTSP(Real Time Streaming Protocol)

RTSP は、IETF によって標準化されたリアルタイム性のあるデータ(ストリーミング)の配布を制御するためのプロトコルであり、再生、停止、記録などの操作ができる。SIP とは違い、RTSP においてはサーバとクライアントとが明確に区別され、データの流れは基本的にサーバからクライアントへの一方向であるが、サーバからクライアントに送付する要求も定義されている。

サーバからクライアントへのデータ転送には通常 RTP が使用されるが、それに限定されてはいない。RTSP によって制御されるデータの流れに関しては RTSP セッションが存在するが、RTSP も HTTP に似せてある。RTSP の下位のプロトコルとしては、SIP と違って TCP のように高信頼なプロトコルを使用が前提とされている。ただし RTSP の改訂にあ

たっては低信頼なプロトコルを使用できるようにする方向性が示されている。図 2.8 に RTSP Method、図 2.9 に RTSP を使用したサーバ-クライアントのやり取りを示す。

メソッド	方向	要求条件	内容
DESCRIBE	C->S	推奨	セッション情報の取得 (SDP 等)
ANNOUNCE	C->S, S->C	オプショナル	C->S: クライアントからのセッション情報の通知 S->C: セッション情報の更新
GET_PARAMETER	C->S, S->C	オプショナル	セッションパラメータの取得
OPTIONS	C->S, S->C	必須	オプション機能のチェック
PAUSE	C->S	推奨	メディア転送の中断
PLAY	**C->S**	**必須**	**メディア転送の開始、再開**
RECORD	C->S	オプショナル	メディア情報の記録
REDIRECT	S->C	オプショナル	リダイレクション
SETUP	**C->S**	**必須**	**セッションの初期化**
SET_PARAMETER	C->S, S->C	オプショナル	セッションパラメータの設定
TEARDOWN	C->S	必須	セッションの終了

図 2.8 RTSP Method([2]より)

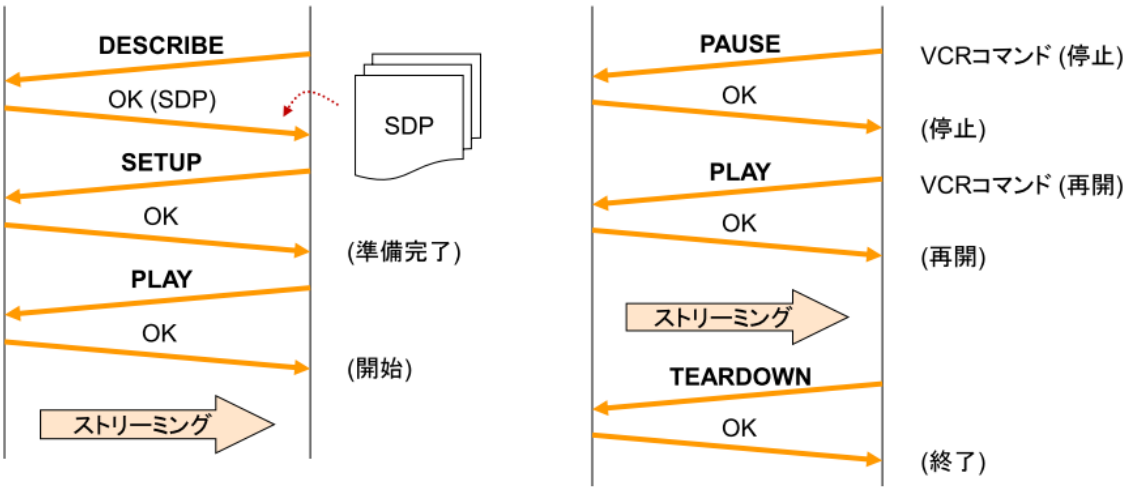


図 2.9 サーバ-クライアント間の通信の様子([2]より)

2.2.4.HTTP Live Streaming

HTTP Live Streaming は Apple が開発した HTTP ベースのストリーミング配信プロトコルである。iOS3.0/Android4.0 以降を搭載しているデバイスで利用可能である。アーキテクチャは図 2.10 のように"Server"/"Distribution"/"Client"の 3 つで構成されている。ま

ず、”Server”コンポーネントで元素材を MPEG-2TS 形式にエンコードし、ストリームセグメンタと呼ばれるツールを使って細かく分割されたメディアセグメントファイル(一連の低容量 TS ファイル)とインデックスファイルを(M3U8)を生成する。そのファイル群を”Distribution”コンポーネント(一般的な Web サーバ)に展開する。ここまでで配信準備は完了し、HTTP Live Streaming に対応している”Client”ソフトウェア(動画再生プレーヤー)が”Distribution”に設置されたインデックスファイルにアクセスすれば配信が開始される。HTTP Live Streaming は RTSP や MMS などのストリーミング用プロトコルではなく HTTP を使うため、特別なストリーミングサーバを必要とせず、Apache や Nginx などの一般的な Web サーバで動画像を配信することができる。

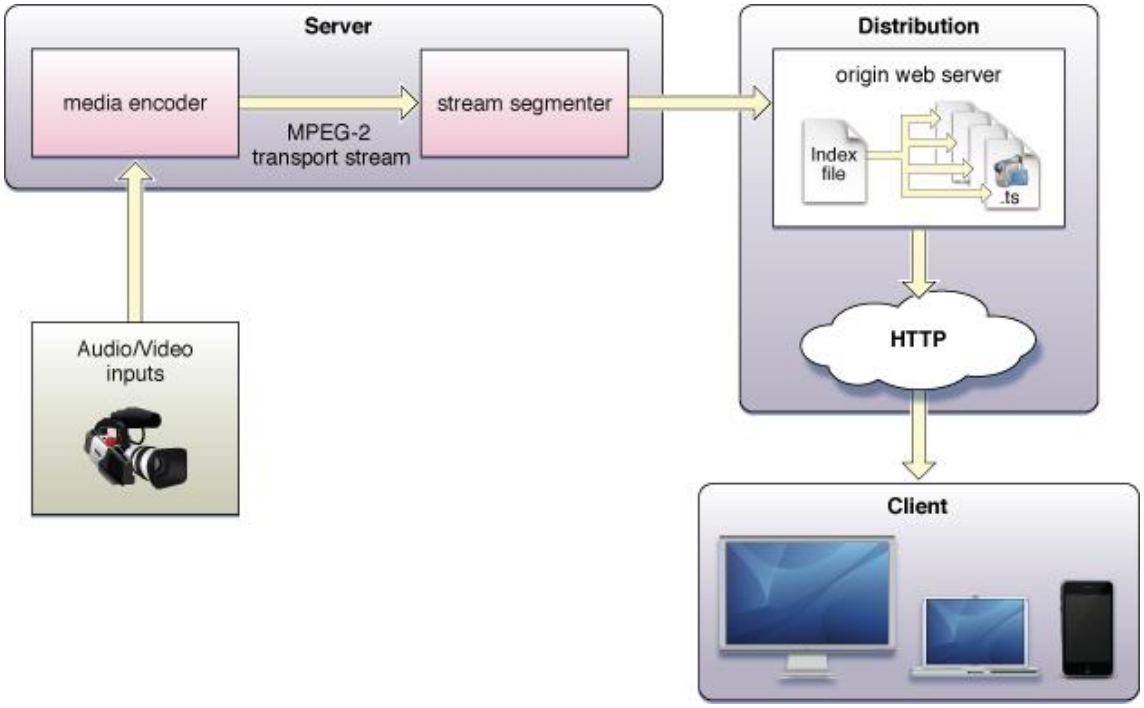


図 2.10 HTTP Live Streaming のアーキテクチャ概要([5]より)

2.2.4.1. サーバコンポーネント

サーバはメディアエンコーダと、エンコードされたメディアをセグメントに分割し、ファイルとして保存する。図 2.10 の場合、メディアエンコーダはオーディオ/ビデオデバイスからリアルタイムな信号を受け取り、転送ストリーム用にカプセル化する。その際のコーデックには H.264 や HE-ACC といったようなクライアントがサポートしている形式を使用する必要がある。また、転送ストリームとしては MPEG2-TS が用いられている。エンコーダはエンコードされたメディアを MPEG2-TS 形式にカプセル化し、ローカルネットワークを通してストリームセグメンタへ転送する。ストリームセグメンタは、ローカルネットワークからトランスポートストリームを読み込むと、等時間の一連の低用量メディアファイルに分割する。また、これはプロセス、通常はソフトウェアである。セグメンタが新しい

メディアファイルを終了するたびにインデックスファイルは更新される。メディアファイルをサポートされるコーデックを使用してエンコードしている場合は、ファイルセグメンタを使用して MPEG-2 トランスポートストリームにカプセル化し、同じ長さのセグメントに分割する。インデックスファイルは通常、ストリームセグメンタまたはファイルセグメンタにより生成され、MP3play リストに使用される.m3u 形式の拡張形式である.M3U8play リストとして保存される。図 2.11 に実際のインデックスファイルの一例を示す。

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:13
#EXTINF:10.416667,
bbb8000_60s/stream000.ts
#EXTINF:12.625000,
bbb8000_60s/stream001.ts
#EXTINF:10.416667,
bbb8000_60s/stream002.ts
#EXTINF:10.416667,
bbb8000_60s/stream003.ts
#EXTINF:12.208333,
bbb8000_60s/stream004.ts
#EXTINF:3.916667,
bbb8000_60s/stream005.ts
#EXT-X-ENDLIST
```

図 2.11 インデックスファイルの一例

2.2.4.2. ディストリビューションコンポーネント

ディストリビューションシステムは、メディアファイルとインデックスファイルを HTTP 通信を利用してクライアントへ配信する、ウェブサーバかウェブキャッシュシステムである。表 2.1 のように、.M3U8 ファイルと.ts ファイルの MIME タイプの関連付けを行う。

表 2.1 HTTP Live Streaming の MIME タイプ設定

ファイル拡張子	MIME タイプ
.M3U8	Application/x-mpegURL または Vnd.apple.mpegURL
.ts	Video/MP2T

2.2.4.3. クライアントコンポーネント

クライアントソフトウェアは起動後、ストリームを識別する URL に基づき、インデックスファイルを取得する。インデックスファイルはこの後、使用可能なメディアファイル、復号キー、利用可能なすべての代替ストリームの場所を指定する。選択されたストリームに対して、クライアントは利用可能なメディアアイルを順番にダウンロードする。各ファイルには、ストリームの連続したセグメントを含む。十分なデータがダウンロードされると、クライアントはデコードされたストリームのユーザへの表示(再生)を開始する。このプロセスは、クライアントがインデックスファイルで#EXT-X-ENDLIST タグを検出するまで続き、#EXT-X-ENDLIST タグが存在しない場合、インデックスファイルは配信中の放送の一部である。

2.2.5.MPEG Dynamic and Adaptive Streaming over HTTP

MPEG-DASH の特徴は、スマートフォンやタブレット端末、テレビなどの様々な機器に対して、動画が途切れないように帯域などの環境に応じてビットレートを切り替えることで、最適な高品質ビデオストリームを提供することを目的としている。音声や字幕を多言語で配信し、リモコンなどの操作で言語を切り替えて視聴することなどができることも特徴である。動画ストリーミング配信には、RTSP や RTMP を使わず HTTP プロトコルを利用している。通常の WEB サイト同様に CDN(Contents Delivery Network)を利用することができ、配信コストを低減できる点も特徴である。ISO/IEC で標準規格として勧告化された[ISO/IEC 23009-1 MPEG-DASH パート 1]では、MPEG-DASH クライアントが利用するストリーミング再生に必要な動画ファイルの情報が記載されたメタデータである MPD(The Media Presentation Description)仕様と動画ファイルのセグメント形式について記載されている。図 2.12 に MPEG-DASH のアーキテクチャを示す。

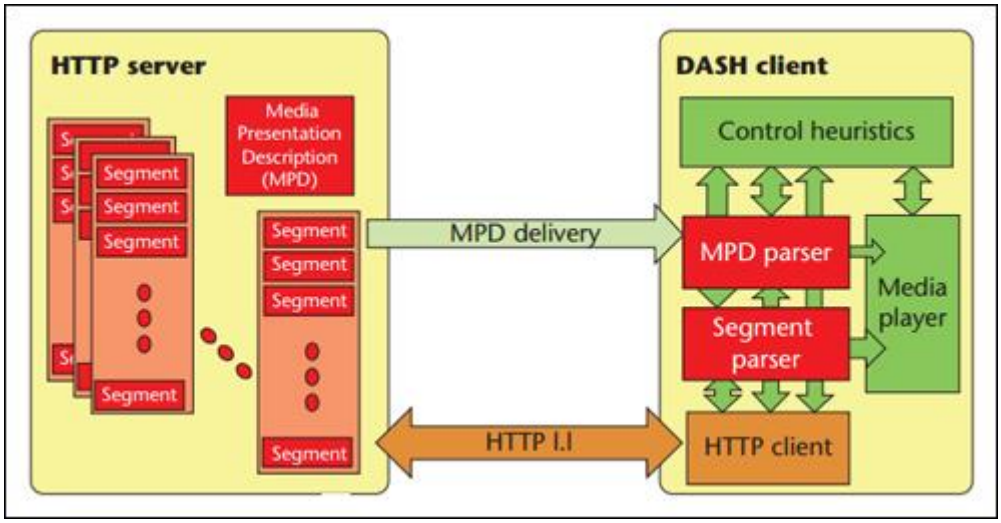


図 2.12 MPEG-DASH のアーキテクチャ([6]より)

2.2.5.1. MPD(The Media Presentation Description)

MPD ファイルは、XML 形式の階層構造で、メディアファイル情報が記載される。MPD ファイルは、番組やコンテンツを構成する 1 つ単位の「Period」、映像・音声・字幕などの符号化関連情報を記述する「AdaptationSet」、映像・音声のビットレートや解像度、アスペクト比などの情報を記述する「Representaion」、メディアファイルを分割したセグメントに関する情報を記述する「segmentList」で構成される。なお、MPD ファイルのプロファイルの定義や配信するメディアファイルのストリーミング種別、メディアファイルの種類、長さ、MPD ファイルの場所などの情報は MPD ファイルの先頭に記載する。図 2.13 に MPD ファイルの構成、図 2.14 に MPD ファイルの例を示す。

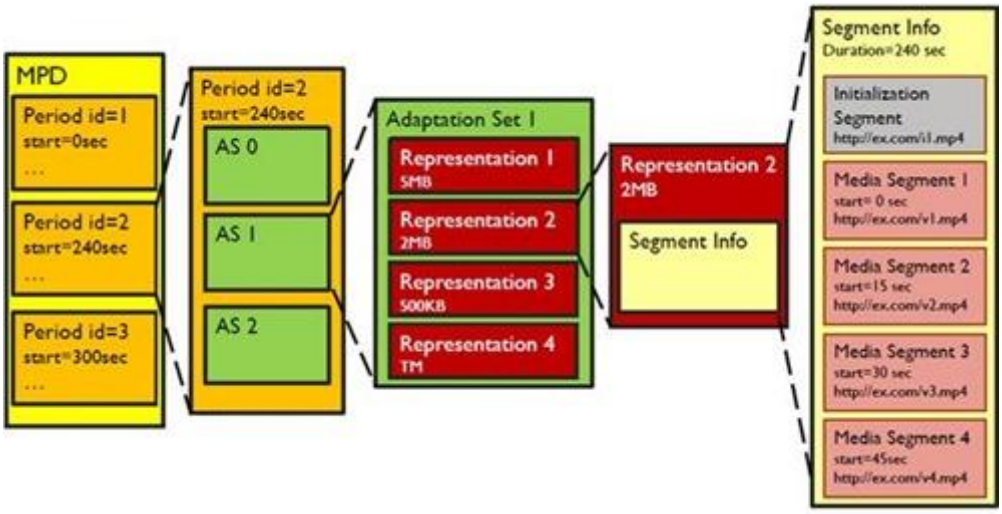


図 2.13 MPD ファイルの構成([4]より)

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="urn:mpeg:mpegB:schema:DASH:MPD:DIS2011"
xsi:schemaLocation="urn:mpeg:mpegB:schema:DASH:MPD:DIS2011"
profiles="urn:mpeg:mpegB:profile:dash:isoff-basic-on-demand:cm"
type="OnDemand"
mediaPresentationDuration="PT0H9M56.46S" minBufferTime="PT10.00S">
<Period>
<Group segmentAlignmentFlag="true" mimeType="video/mp4">
<Representation mimeType="video/mp4" width="1280" height="720" startWithRAP="true"
bandwidth="769540" minBufferTime="15000">
<SegmentInfo duration="PT15.00S">
<InitialisationSegmentURL sourceURL="bbb_900kbps/bunny_900kbit_dash.mp4"/>
<Url sourceURL="bbb_900kbps/bunny_15s1.m4s"/>
<Url sourceURL="bbb_900kbps/bunny_15s2.m4s"/>
<Url sourceURL="bbb_900kbps/bunny_15s3.m4s"/>
<Url sourceURL="bbb_900kbps/bunny_15s4.m4s"/>
<Url sourceURL="bbb_900kbps/bunny_15s5.m4s"/>
</SegmentInfo>
</Representation>
</Group>
</Period>
</MPD>
```

図 2.14 MPD ファイルの一例

A) Presentation

MPD の階層構造のトップレベル。Presentation は完結した 1 本のビデオストリームを抽象化したものであり、タイトルや再生時間などは定義されるが、言語や解像度やビットレートなどは不定で。Period の並びから構成される。

B) Period

Period はビデオを時間で分割したものであり、start 時刻と duration 時間が定義される。両者が計算できればどちらか一方でもよい。Group から構成され、Group は Representation から構成される。

C) Representation

Representation は Segment から構成される。

D) Segment

Segment はメディアの最小単位であり、HTTP range GET で取得できるようにバイト位置や時刻情報などが記述されている。

2.2.6.DASHEncoder

DASHEncoder とは、[7]にて公開されており、DASH コンテンツを生成することを可能とするオープンソースアプリケーションである。DASHEncoder は DASHEncoder.config という設定ファイルでパラメータを指定することによりエンコード、セグメンテーションを行い、さらに MPD ファイルを生成する。図 2.15 に DASHEncoder.config の一例を示す。

```
#====================================
# General Options
#====================================

dest-directory : /var/www/test/
video-encoder : x264
audio-encoder : ffmpegAAC
multiplexer : mp4box
input-res : 1920x1080
#====================================
# x264 Options
#====================================

bitrate : 200|400|600|1000|2000|3000
statistics : stat.temp
gop : 48
scenecut : 0
profile : baseline
preset : slow
input : /home/yuya/videos/big_buck_bunny_1080p24.y4m

passes : 1
const-filesize : 0

#Additional Options for Encoding
pass1 : --verbose --fps 24
pass2 : --verbose --psnr
#====================================
# MPD Options
#====================================
mpd-name : bbb1080p.mpd
url-root : http://localhost/video/
#set-base-url
mpdActStandardPostfix :  actVersion.mpd
duration : 9M56.46S
transform-mpd
minBufferTime : 5.0S
```

図 2.15 DASHEncoder.config の一例

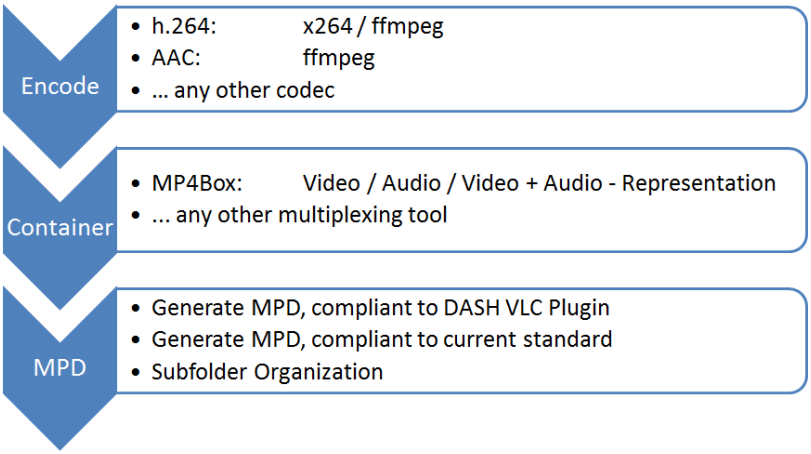


図 2.16 DASHEncoder の DASH コンテンツ生成のプロセス([7]より)

2.2.7.DASH-JS

DASH-JS は、HTML5 の video エレメントを利用した MPEG-DASH をウェブ用にシームレスに統合した JavaScript により記述されたアプリケーションである。また、Google Chrome の Media Source API を利用し、様々な端末での DASH 配信を可能とし、簡単にウェブアプリケーションを作成することができる。図 2.17 に DASH-JS のアーキテクチャを示す。EventHandlers は Media Source API とのインターフェースとして働き、Media Source API からのイベントを受け取り、それぞれに対する処理を行う。EventHundler は Media Source API からイベントを受け取ると、MPD ファイルをダウンロードする。MPD ファイルは MPD Parser により、それぞれのレプレゼンテーション、ビットレート、解像度といった情報を解析する。これらの情報はレプレゼンテーションの決定や、MPEG-DASH のアダプテーションロジックに利用される。Segment Requester は xmlHTTPRequest を使用して HTTP リクエストを生成し、非同期、同期の HTTP リクエスト、HTTP 応答を実現している。また、Bandwidth Estimator はスループットの有効スループットの測定、推測を行う。セグメント番号 n 番目の時点でのスループット b_n は式(2.1)によって計算される。b_{n-1} は n-1 番目のセグメント受信時のスループットを表し、b_m は n-1 番目のセグメントを受信することにより測定したスループットを表す。また、w_1、w_2 は重み係数である。この 2 つの重み係数により、観測されたスループットと、一つ前に観測されたスループットの重みを適切なものとすることができる。続いて、Base Buffer クラスは再生バッファ機能を提供している。また、バッファのサイズ指定の方法として、「時間」と「バイト」の 2 種類を提供している。指定されたクリティカルバッファサイズ(最小バッファサイズ)までバッファが減少した場合、次のセグメントを最大バッファサイズに達するまで要求する機能を有している。また、バッファと推測された帯域は、バッファの状況や帯域によって選択するレプレゼンテーションを決定するために使用される。複数のアダプテーションロジックを実装するために Basic Adaptation Logic クラスが提供されており、すべてのアダプテーショ

ンロジックはこのクラスを継承しなければならない。これはセグメントがダウンロードされる毎に呼ばれる。DASH-JS では Rate Based Adaptation Logic を採用しており、式(2.1)により推測された帯域をベースに、switchRepresentation()メソッドを呼び出し、レプレゼンテーションのアダプテーションを行う。

$$b_n = \frac{w_1 b_{n-1} + w_2 b_m}{w_1 + w_2} \tag{2.1}$$

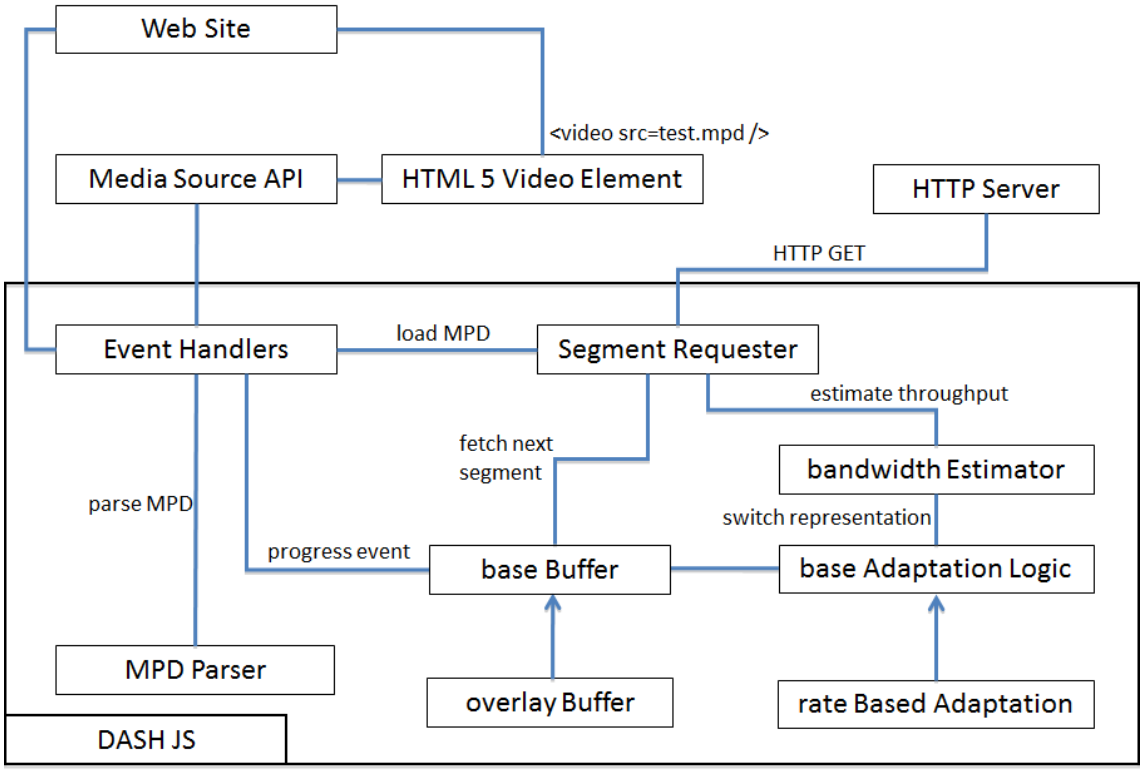


図 2.17 DASH-JS のアーキテクチャ([8]より)

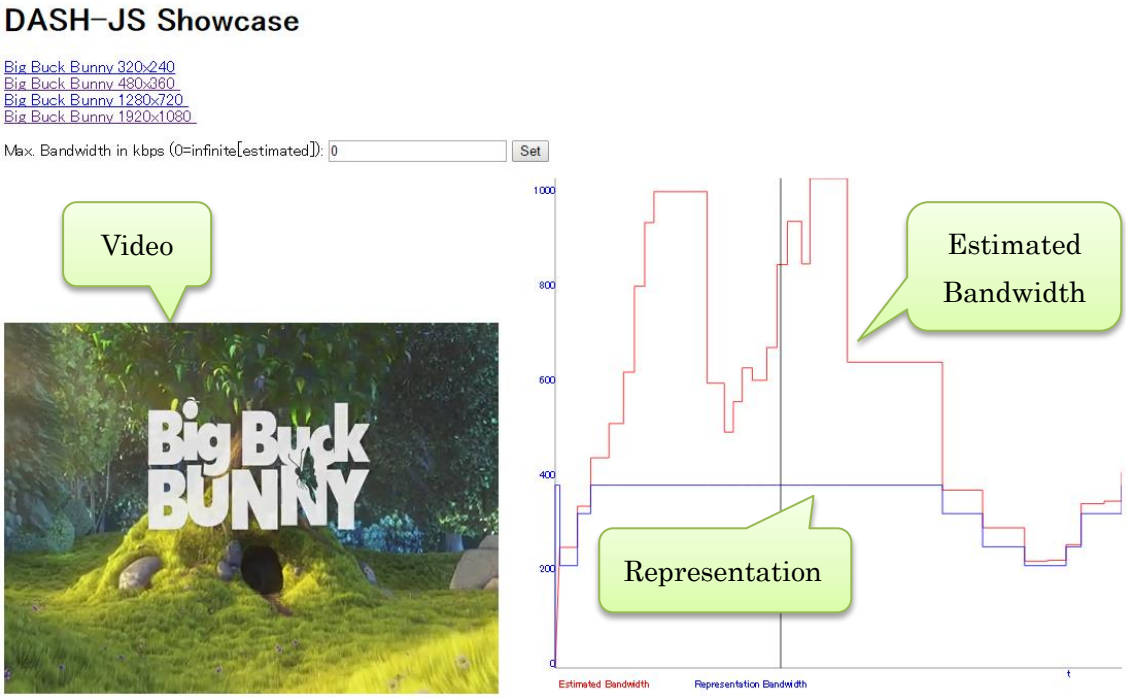


図 2.18 DASH-JS を用いた DASH 配信の様子

第 3 章 NDN(Named Data Networking)

現行のインターネットでは、ユーザが所望のコンテンツを要求する際、コンテンツの場所情報(i.e. URL)を指定することにより、取得することができる end-to-end 通信である。これは、従来のインターネットの使用用途はコンピュータ間の通信を実現するためのものであり、指定された相手にサービスや処理を提供するというものであったためである。しかし近年出現したクラウドサービスなどに見られるように、ネットワークを含めたサービス自体の抽象化の概念が普及してきており、人々は具体的にネットワーク上のどのノードがサービスを提供しているかということを意識せずにサービスを利用している。つまり、現在のインターネットでは「サービス」、あるいは「モノ」を意識することにより通信を行っている。このインターネットのアーキテクチャと人々の利用用途の乖離を解決するために提案されたのが、コンテンツ指向型ネットワーク CCN(Content Centric Networking)[9]である。この CCN は、現行の IP ネットワークの「場所」の概念をそのまま「モノ」の概念に置き換えたものである。このネットワークでは、ユーザは所望のコンテンツの ID を宣言するだけで通信を実現することができる。また、ネットワーク内のキャッシングメカニズムにより、無駄なトラフィックの削減や低レイテンシを実現することが期待されている。NDN[10]はこの CCN をベースに研究開発されており、現行のネットワークに置き換わる次世代ネットワークとして期待されている。

3.1. NDN における通信手法の概要

NDN における通信はデータの受信者主導で行われ、*Interest* と *Data* と呼ばれる 2 つのパケットの交換により行われる。この両者のパケットはコンテンツをさらに分割したチャンクを識別する name を含んでいる。データの要求者(Consumer)は Interest パケットに所望のデータのチャンクの name を付与し、ネットワークへブロードキャストする。ルーターはこの name を利用して Interest パケットを、所望のデータを持つノード(Producer)までフォワーディングする。Interest パケットが所望のデータを持つノードへ到達すると、ノードは name、所望のデータ、name とデータを紐づける署名等を含んだ Data パケットを送り返す。この Data パケットは、Interest パケットが通ったパスを戻り、Consumer まで送り届けられる。図 3.1 に Interest パケットと Data パケットの構成を示す。

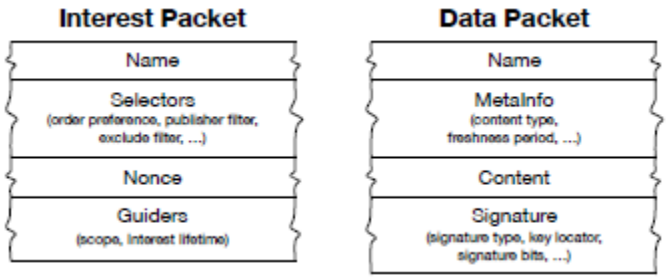


図 3.1 Interest パケットと Data パケットの構成([10]より)

3.2. NDN ノードの構成

Interest パケットと Data パケットのフォワーディング機能を実現するため、NDN ノードには以下の 3 つの機能が実装されている。

3.2.1. PIT(Pending Interest Table)

フォワーディングし、Data パケットを Consumer へ届け終わっていないすべての Interest パケットを記録する。また、PIT はインターネットから流れてきた name を受信したインターフェース､送信したインターフェースと共に記録する｡NDN ノードは Interest パケットを受信すると、初めに CS をに name が一致するデータが存在するか確認するが、存在せず、かつ PIT に name が一致する情報が存在する場合、その Interest パケットを受信したインターフェースの PIT への登録のみを行う。ノードが Data パケットを受信すると PIT から一致する情報を検索し、それを基に Interest パケットの受信インターフェースへ Data パケットを送信し、PIT の情報を削除する。

3.2.2. FIB(Forwarding Information Base)

Interest パケットが到着し、CS にデータが存在せず、PIT にも一致する情報が存在しない場合、この FIB を基づいて Interest パケットのフォワーディングを行う。

3.2.3. CS(Content Store)

Data パケットがノードを通過した場合、CS をキャッシングされる。通過したデータをすべてキャッシングせず、LRU(Least Recently Used)や LFU(Least Frequently Used)といったキャッシングアルゴリズムが実装されている。

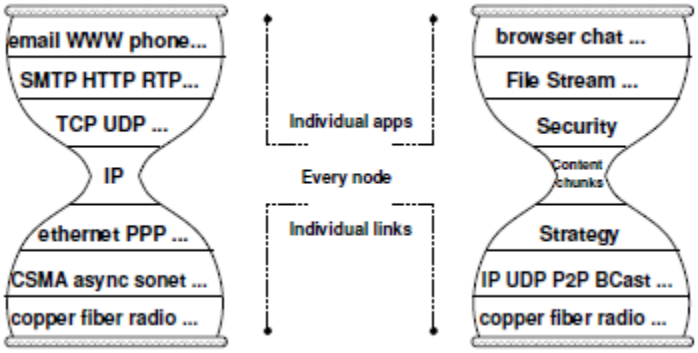


図 3.2 IP と NDN のプロトコルスタックの関係([9]より)

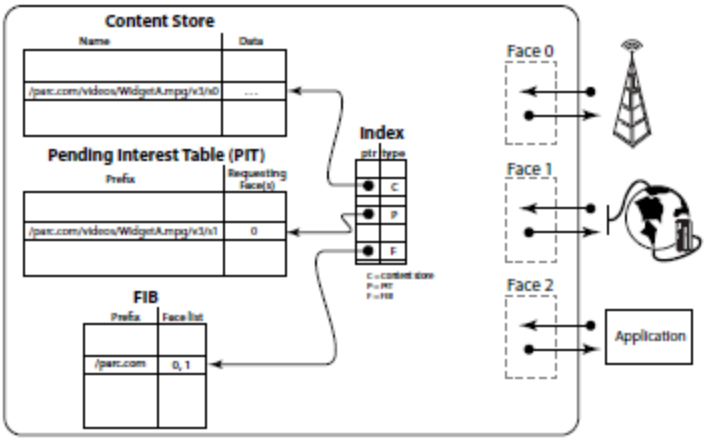


図 3.3 NDN ノードのフォワーディング機能([9]より)

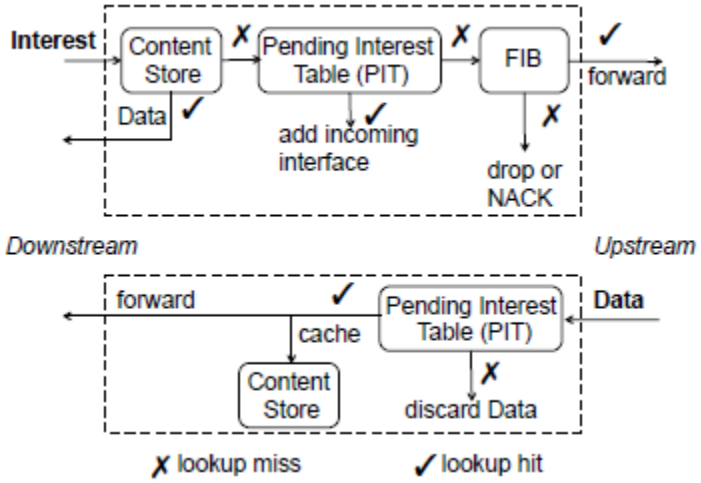


図 3.4 NDN におけるフォワーディングプロセス([10]より)

3.3. NDN オープンソースアプリケーション

様々な NDN の課題を持つ NDN アーキテクチャの開発、展開を促すため、いくつかのオープンソースアプリケーションが存在する。

3.3.1. NDNx/CCNx

CCNx[11]は、CCN のコミュニケーションアーキテクチャの発展、促進、評価を目的として設立された Project CCNx によって提供されているオープンソースアプリケーションである。現行のネットワークのオーバーレイコミュニケーションを実現することが可能なミドルウェアソフトウェアである。マシンにインストールし、デーモンを実行することにより、簡単に CCN 環境を模擬することができる。また、端末間の通信には UDP、TCP の両者をサポートしている。

NDNx[12]は、CCNx のフォーク版であり、NDN Project により NDN アーキテクチャの研究開発を目的として開発されている。NDNx version 0.1 は CCNx version 0.7.2 のフォーク版であり、プログラム中のすべての”ccn”が”ndn”と書き換えられたものとなっている。

3.3.2. NDN-TLV

NDN Project は NDN とフォワーディングを行うコードベースの両者へのパケットフォーマットへの移行を行った。NDN-TLV パケットの形式は Cisco 社の NDN プロジェクトチームが提案した形式を採用したものである。NDN パケットは、Type-Length-Value(TLV)と 3 種類の要素へとエンコードされる。NDN の Interest パケットと Data パケットの区別は Type 部分により区別される。また、NDN のパケットは主に TLV 形式のパケットがネストされた構造を取っており、TLV 層の順序をそろえたり、処理のオーバーヘッドやエラーが少なくなるようにネストの数を制御する。NDN のパケットフォーマットのヘッダーのサイズは固定ではなく、パケットサイズによるオーバーヘッドの削減や、パケットサイズを小さく保つことへ貢献している。

3.3.3. NDN-JS

NDN-JS はJavaScriptにより記述されたクライアントサイドのNDNライブラリである。NDN-JS を用いることにより、CCNx/NDNx パッケージをインストールすることなく、HTML5 と JavaScript をサポートしているブラウザを用いて NDN プロトコルを利用して通信を実現することができる。

3.3.3.1. NDN-JS を利用したコンテンツダウンローディング

従来の同期式の GET/POST、非同期の XMLHttpRequest などの HTTP ベースのメソッ

ドは NDN にとって適切ではない。これは、NDN にはサーバにプッシュするという概念が存在しないからである。そこで、NDN-JS では WebScoket を利用して NDN パケットの交換を実現している。しかし、従来の CCNx/NDNx ルータは WebSocket プロトコルをサポートしていないため、NDN-JS とコネクションを張り、受け取ったパケットを NDN ルータへ転送することを可能とした WebSocket proxy を用いている。ここで、以下に NDN-JS を利用したコンテンツダウンローディングの一例を示す。図 3.5 はクライアントウェブブラウザ、WebSocket proxy、NDN ルータの関係を表している。クライアントのブラウザは WebSocket proxy に対して WebSocket コネクションを張り、その後 WebSocket proxy は NDN ルータに TCP/UDP コネクションを張る。WebSocket proxy は、WebSocket フレームを受け取ると、その中から NDN パケットを抽出し、NDN ルータへフォワーディングする。また、WebSocket proxy は、NDN パケットが一つの WebSocket フレームへ入らない場合のフラグメンテーションや再アセンブルの役割も担っている。さらに、WebSocket proxy は Node.js[14]を利用して駆動しており、ポート番号は 9696 を利用している。NDN-JS から送信されるパケットは TLV パケットであるため、NDN ルータで起動するデーモンは NDNx-TLV を利用する。

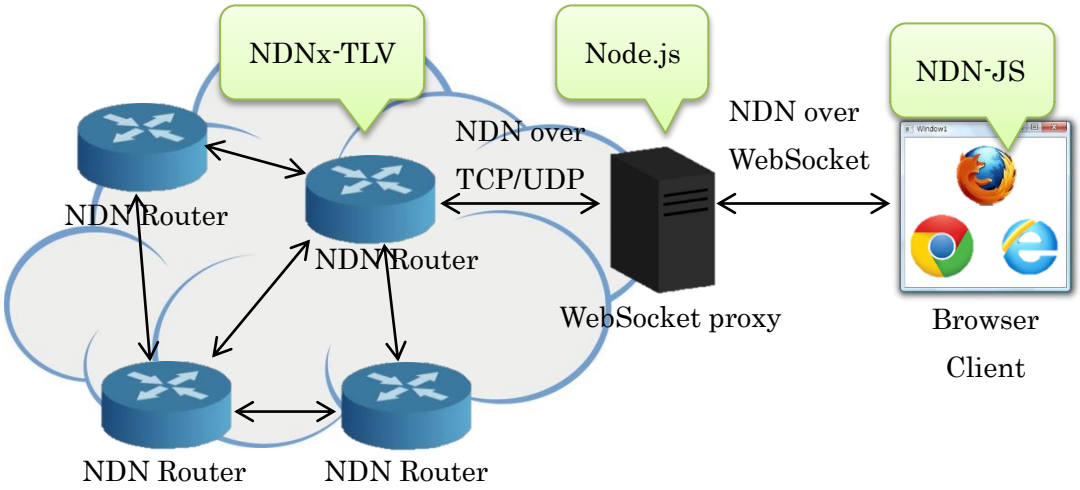


図 3.5 WebSocket proxy を利用したブラウザの NDN への接続

以下に tearsofsteek4k_3000kbit/tearsofsteel4k1.m4s というコンテンツをダウンロードする際の手順を記載する。

① NDN ルータの設定

NDN ルータでは、NDNx-TLV のデーモンである ndnd-tlv を起動させ、リポジトリを作成し、コンテンツをリポジトリに登録する。

```
$ ndnd-tlv-start   //ndnd-tlv デーモンの起動
$ ndnr   //リポジトリの生成
$ ndnputfile ndn:/tearsofsteek4k1.m4s tearsofsteel4k1.m4s
   //リポジトリへコンテンツの登録
$ node wsproxy-tcp.js //ndn-js/wsproxy/wsproxy-tcp.js
//WebSocket proxy の起動
```

② NDN-JS を利用したコード

```
<?xml version = "1.0" encoding="utf-8" ?>
<html xmlns = "http://www.w3.org/1999/xhtml">
<meta charset="UTF-8">

<head>
  <title>NDN Get File via WebSocket</title>
  <script type="text/javascript" src="ndn-js/build/ndn.js"></script>

  <script type="text/javascript">
    hostip = "192.168.1.2";   //WebSocket proxyのIPアドレスを指定
    var face = new Face({port:9696,host:hostip});  //WebSocket proxyへコネクションを張る

    //Dataパケットを受信した際に呼び出される関数
    function onData(interest, data) {
      document.getElementById('content').innerHTML += "Size: " + data.getContent().size() + " [KB]";
    }
    //Interest timeoutが起きた場合に呼び出される関数
    function onTimeout(interest) {
      face.expressInterest(interest, onData, onTimeout);
    }

    function run() {
      var name = new Name(document.getElementById('interest').value);
      face.expressInterest(name, onData, onTimeout);
    }
  </script>

</head>
<body >
  <form>
    Please Enter an Interest:<br />
    <input id="interest" type="text" name="INTEREST" size="50" value="ndn:/1MB" />
  </form>
  <button onclick="run()">Fetch Content</button>
  <p id="content"><br/></p>
</body>
</html>
```

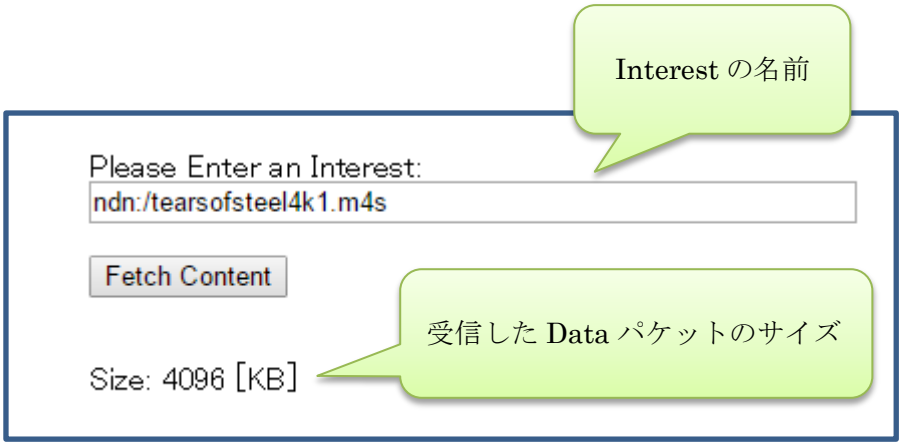


図 3.6 NDN-JS を利用した Data パケットダウンロードの様子

図 6 のように Interest パケットの名前を指定し、「Fetch Content」ボタンを押すことでブラウザを使用して NDN プロトコルで通信することができる。

第 4 章 TCP の輻輳制御方式

この章では、TCP(Transmission Control Protocol)の輻輳制御について述べる。TCP の輻輳制御方式には大きく分けてロスベース方式と遅延ベース方式に分けられる。ロスベース方式では、エンドノード側は通信経路の状況を把握できないので、輻輳が起こらずパケットが損失した場合はネットワークに余裕があると判断し、転送速度を上げる。反対に輻輳が起こりパケットが損失した場合はネットワークに余裕がないと判断し、転送速度を下げる。また、遅延ベース方式では、RTT(Round Trip Time)の測定値と推測値を用いて制御を行う。測定値が推測値を下回ればネットワークが空いていると判断し、転送速度を上げる、反対に測定値が推測値を上回ればネットワークが混んでいると判断し、転送速度を下げる。また、この方式ではパケットロスが起こる前に適切な転送速度へ制御するため、ロスベース方式よりパケットロスが少なく、また帯域を有効に活用することが可能である。本章では、TCP Tahoe、TCP Reno、TCP Vegas、Cubic TCP について述べる。

4.1. TCP Tahoe

TCP Tahoe は 1988 年頃から利用されている、スロースタート状態と輻輳回避状態が存在し、ウィンドウ制御と Fast Retransmit という高速再送アルゴリズムを採用しており、ウィンドウサイズは式(4.1)、式(4.2)のように変化する。

$$cwnd = cwnd + 1 \qquad cwnd < ssthresh \tag{4.1}$$

$$cwnd = cwnd + \frac{1}{cwnd} \qquad cwnd > ssthresh \tag{4.2}$$

また、閾値(ssthresh)を定めるために輻輳が起きてパケットロスが起きるまでウィンドウサイズを上げ続ける。輻輳が起こりパケットロスが起こると、式(4.3)のようにその時点でのウィンドウサイズの半分(cwnd/2)を ssthresh と定める。

$$ssthresh = \frac{cwnd}{2} \tag{4.3}$$

$$cwnd = 1 \tag{4.4}$$

以降は ssthresh までスロースタート状態であり、ACK を受信する毎に 1 ずつウィンドウサイズを増加させ、stthresh 以上になると輻輳回避状態へ移行し、ウィンドウサイズ分の 1(1/cwnd)ずつ線形的に輻輳ウィンドウサイズを増加させる。この輻輳回避状態で再びパケットロスが起こった場合、式(4.4)のようにウィンドウサイズを 1 まで減少させ、同じ動作を行う。以上の様子を図 3.7 に示す。

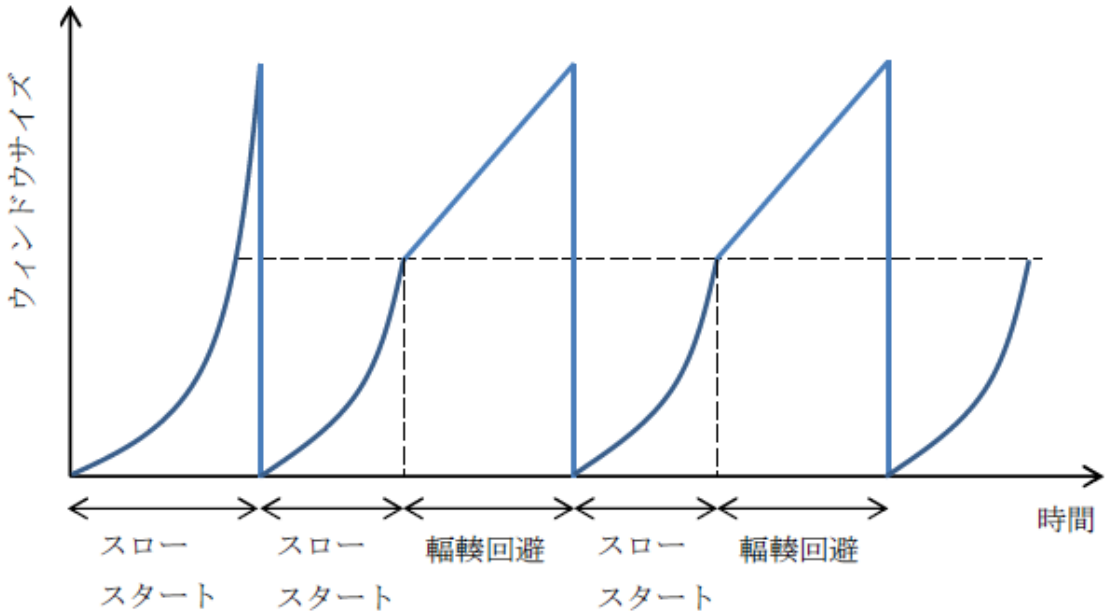


図 3.7 TCP-Tahoe の輻輳ウィンドウの変化の様子

4.2. TCP Reno

TCP Reno は 1990 年に登場し、前述の TCP Tahoe に改良を加えた Fast Recovery アルゴリズムを採用している。Tahoe の場合、輻輳が起きパケットロスが起こるとウィンドウサイズを 1 まで減少させ、スロースタート状態から開始するため通信速度を極端に落としてしまうという問題が存在した。そこで TCP Reno では、のようにパケットロスが起きた時点のウィンドウサイズの半分を閾値とし、式(4.5)のように後にパケットロスが発生した際はウィンドウサイズを 1 まで下げずに閾値まで下げ、輻輳回避状態へ移行する Fast Recovery アルゴリズムを採用することにより過剰なウィンドウサイズの減少を回避し、高速な転送を可能としている。

$$cwnd = ssthresh \tag{4.5}$$

図 3.8 に TCP Reno の輻輳ウィンドウの変化の様子を示す。

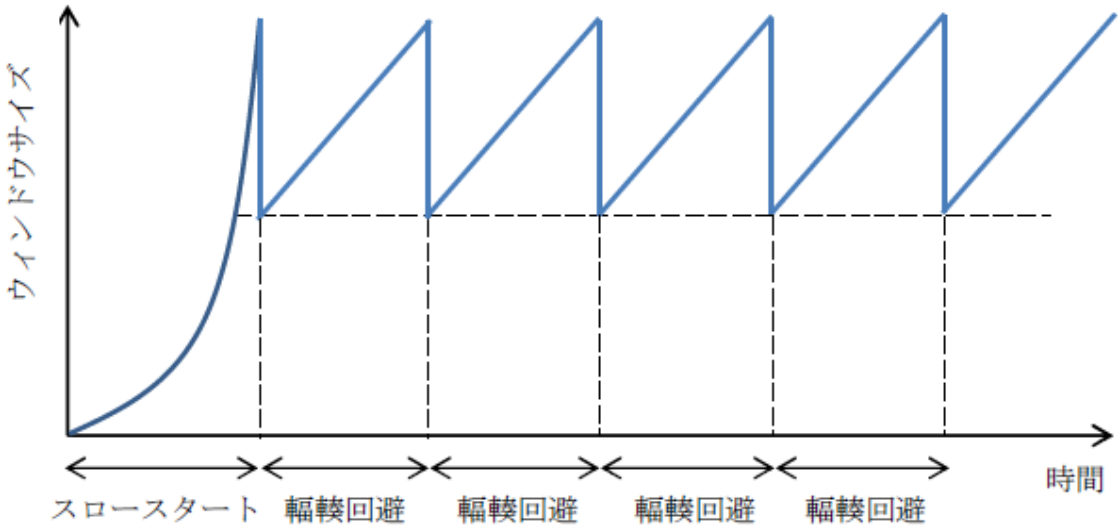


図 3.8 TCP Reno の輻輳ウィンドウの変化の様子

4.3. TCP Vegas

TCP Vegas は Tahoe や Reno と異なり、遅延ベースの輻輳制御方式である。詳細な通信経路の状態を推測する機能があり、混雑の具合に応じて転送速度を制御することができる。通信経路の把握には RTT(Round Trip Time)を利用する。基本的なアルゴリズムとしては、式(4.6)、式(4.7)のように観測されたスループット(Actual Throughput)と推測されたスループット(Expected Throughput)の二つを用い、差分を求めることによって経路中のバッファに溜まっているパケット数を推測しつつウィンドウサイズの制御を行う。

$$(\text{Expected Throughput}) = \frac{cwnd}{RTT_{min}} \tag{4.6}$$

$$(\text{Actual Throughput}) = \frac{cwnd}{RTT} \tag{4.7}$$

$$diff = \left(\frac{cwnd}{RTT_{min}} - \frac{cwnd}{RTT}\right) \tag{4.8}$$

差分 $diff$ を用いて式(4.9)のようにウィンドウ制御を行う。

$$cwnd = \begin{cases} cwnd + 1 & (diff < \alpha) \\ cwnd & otherwise \\ cwnd - 1 & (diff > \beta) \end{cases} \tag{4.9}$$

ここで α と β はそれぞれ定数であり、経路中のバッファに蓄積されていると予測する範囲を表す。$diff$ の値が α より小さい場合、つまり経路中のバッファにパケットがあまり溜まっていない場合は経路が空いていると判断し、ウィンドウサイズを 1 増加させる。また $diff$ の値が β より大きい場合、つまり経路中のバッファにパケットがたまっている場合は経路が混んでいると判断し、ウィンドウサイズを 1 減少させる。また、その他の場合は、経路

中のバッファの数が許容できる状態であると判断し、ウィンドウサイズの増減は行わない。図 3.9 に TCP Vegas のウィンドウサイズの挙動の一例を示す。

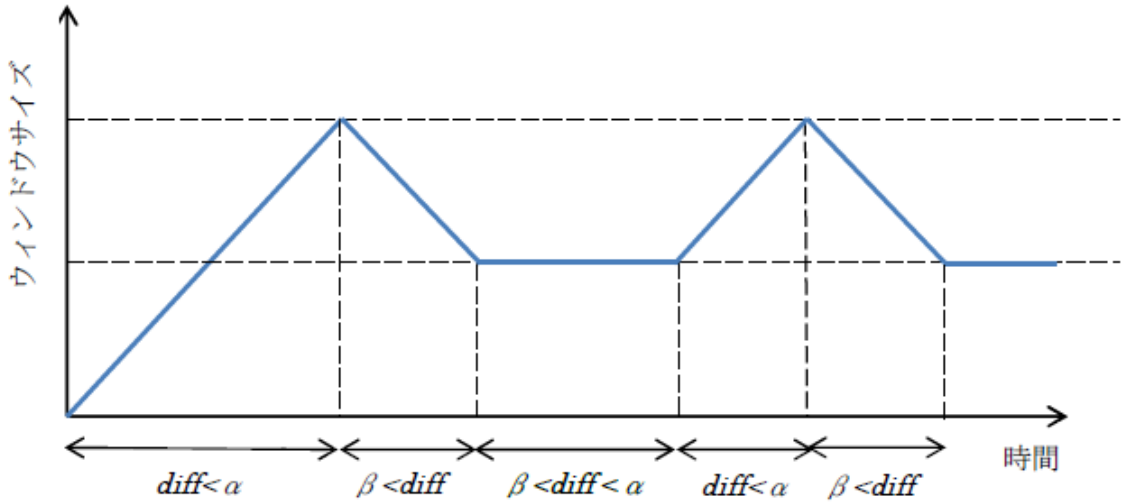


図 3.9 TCP Vegas の輻輳ウィンドウの変化の様子

4.4. CUBIC TCP

CUBIC TCP は、BIC TCP を改良したものであり、高速ネットワーク環境に適した TCP 輻輳制御アルゴリズムである。CUBIC TCP のウィンドウ制御は式(4.10)、式(4.11)に従う。

$$cwnd = C(t-K)^3 + W_{max} \tag{4.10}$$

$$K = \sqrt[3]{\frac{\beta W_{max}}{C}} \tag{4.11}$$

ここで、C、t、および W_{max}はそれぞれ CUBIC 定数、最後にパケットロスが発生した時点からの経過時間、前回パケットロスが発生した時点のウィンドウサイズを表し、βは増加関数を表す。また、パケットロスが起きた場合、式(4.12)に従いウィンドウサイズを減少させる。パケットロスが起きた時点のウィンドウサイズが前回のロス時より小さい場合は W_{max}の値を式(4.13)のように変化させる。

$$cwnd = cwnd \times (1-\beta) \tag{4.12}$$

$$W_{max} = cwnd \times \frac{2-\beta}{2} \tag{4.13}$$

CUBIC TCP の輻輳ウィンドウの変化の様子を図 3.10 に示す。

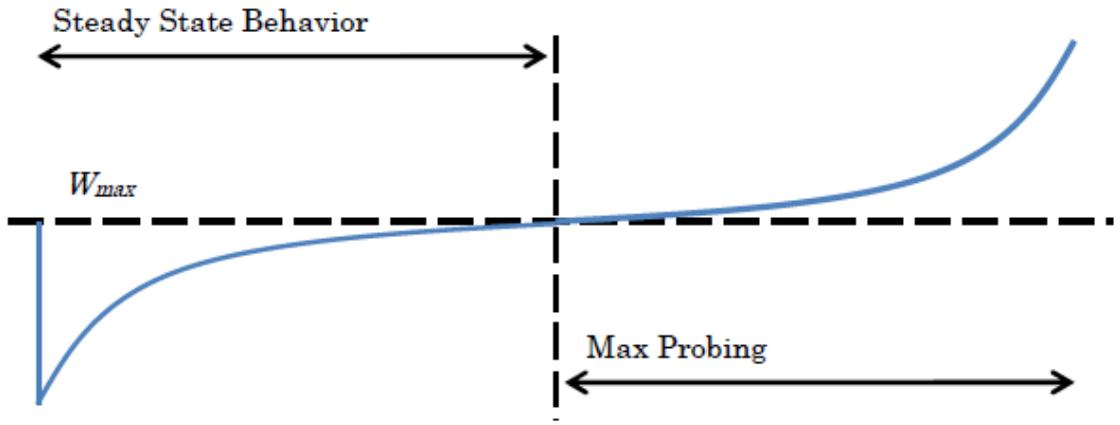


図 3.10 CUBIC TCP の輻輳ウィンドウの変化の様子

パケットロスを検出するとウィンドウサイズを減少させる。また、減少後は速やかに W_{max} に近づくようにウィンドウサイズを増加させる。W_{max} の近傍でウィンドウサイズの増加はほぼ 0 となり、CUBIC はウィンドウサイズをさらに大きくするために Max Probing にフェーズに入り、$W_{cubic} = W_{max}$ に対称的な増加を行う。また、この関数はウィンドウ増加率は経過時間である t に依存しているため、高い RTT 公平性を示す。

第 5 章 NDN における輻輳制御方式性能評価

本章では、NDN における輻輳制御方式について述べる。NDN においても輻輳制御方式が実装されている。しかし、NDN では以下のように第 4 章での述べた IP 上での輻輳制御方式と異なる特徴を持つ。

- ✓ 輻輳制御はサーバ主導ではなく、クライアント主導で行われる。
- ✓ 1 つの Interest パケットは 1 つの Data パケットに対応する。
- ✓ Interest パケットは輻輳ウィンドウのサイズの通知の役割を果たす。

また、NDN はコンテンツ指向型ネットワークであるため、クライアントが受信するコンテンツの出所は異なるノードとなるケースが存在する。

5.1. NDNx の輻輳制御方式

NDN のオープンソースアプリケーションである NDNx における輻輳制御方式について述べる。また、使用するツールとして、NDNx に標準付属する ndncatchunks2 を利用することとする。

NDN ではパケットロスを感知する手法として、InterestLifetime と呼ばれるタイムアウト値が実装されている。これは、Consumer が Interest パケットを送信してから一定時間内に Data パケットを受信しない場合、Interest パケットまたは Data パケットが経路中で廃棄されたと判断し、Interest パケットの再送を行う、という処理であり、デフォルトでは 4[s]に設定されている。NDN のオープンソースアプリケーションである NDNx ver.0.3 の ndncatchunks2 に実装されている輻輳制御方式を以下に示す。

表 5.1 NDNxver0.3 の輻輳制御方式

Algorithm of Congestion Control in NDNx ver0.3.	
1	At UP CALL arrival from the daemon
2	if (it signals an EXPIRED Interest)
3	{cwnd = 1;
4	re-sent the expired Interest}
5	else if (if signals an OUT-OF-ORDER Data)
6	cwnd--;
7	else if (it is ORDER Data && cwnd < MAXIMUM_WINDOW)
8	cwnd++;

ただし、MAXIMUM_WINDOW、InterestLifetime については、

$$\mathrm{MAXIMUM_WINDOW} \leq 31$$

$$\mathrm{InterestLifetime} = 4$$

となっている。

5.1.1. NDNx の RTT 推測機能

輻輳制御方式には利用されていないが、NDNx の ndncatchunks2 には、RTT の推測値を求める機能が実装されている。以下に RTT の推測値 RTTe を求めるアルゴリズムを示す。推測した RTTe より観測した RTT が大きい場合、経路が混んでいると判断し、RTTe を $1/2^3$ の割合で増加させる。また、推測した RTTe より観測した RTT が小さい場合は経路が空いていると判断し、RTTe を $1/2^7$ の割合で減少させる。

```
if (RTT <= 3000000) {
        if(RTT > RTTe)
                RTTe = RTTe + (RTTe >> 3);
        else
                RTTe = RTTe – (RTTe >> 7);
}
```

5.1.2. NDNx の RTT について

NDNx における RTT は、従来の IP における RTT と異なったものとなっている。その様子を図 5.1 に示す。

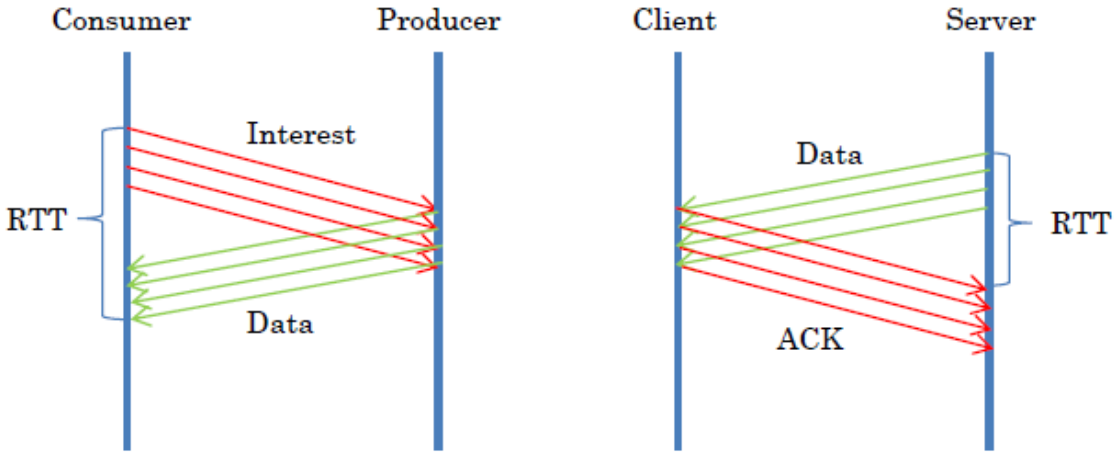


図 5.1 NDNx の RTT と IP の RTT の関係

従来の IP における RTT は、サーバが送った一つのデータを送り、それに対する ACK が返ってくるまでの時間を表す。一方、NDNx における RTT とは、Consumer があるウィンドウサイズ分の Interest パケットを送ってから、ウィンドウ全体に対する Data パケットすべてを受信し終わるまでの時間を指す。

5.2. 提案方式①

NDNx ver0.3.において標準で実装されている輻輳制御方式に対し、3 つの輻輳制御方式の提案を行う。

5.2.1. RTT-Base 方式

以下にアルゴリズムを示す。

$$\mathrm{cwnd(t')} = \begin{cases} \mathrm{cwnd(t)} + 1 \ (\mathrm{rtt} < rtte) \\ \mathrm{cwnd(t)} - 1 \ \ (\mathrm{rtt} > rtte) \end{cases} \tag{5.1}$$

ここで、rtte とは 5.1.1 で述べた推測された RTT を表す。観測された RTT が推測された RTT より小さい場合、経路が空いていると判断しウィンドウサイズを 1 増加させる。反対に観測値が推測値より大きい場合は、経路が混んでいると判断しウィンドウサイズを 1 減少させる。

5.2.2. TCP Vegas 方式

4.3 にて述べた TCP Vegas 方式を NDNx へ応用したアルゴリズムとなっている。ただし、再送メカニズムが IP と NDN では異なり、NDN では InterestLifetime を基準に再送が行われる。

5.2.3. FAST TCP 方式

FAST TCP 方式では、TCP Vegas 方式のようにウィンドウサイズを 1 ずつではなく、期待値と測定値の差の大きさによって増減させる大きさを変化させている、TCP Vegas 方式の改善版と言える。これによりネットワークの状態の変動への追従性が向上している。以下にアルゴリズムを示す。

$$\begin{cases} \mathrm{w} \leftarrow \min\left\{2\mathrm{w}, (1-\gamma)\mathrm{w} + \gamma\left(\dfrac{\mathrm{baseRTT}}{\mathrm{RTT}}\mathrm{w} + \alpha\right)\right\} \\ \mathrm{w} = 31 \ \ \left(\min\left\{2\mathrm{w}, (1-\gamma)\mathrm{w} + \gamma\left(\dfrac{\mathrm{baseRTT}}{\mathrm{RTT}}\mathrm{w} + \alpha\right)\right\} > 31\right) \end{cases} \tag{5.2}$$

ここで、w、α、γ はそれぞれウィンドウサイズ、定数、0~1 の間の定数を示す。

5.3. 評価実験

5.2 で述べた 3 つの輻輳制御方式を NDNx に実装し、評価実験を行う。

5.3.1. 実験環境

本実験環境について述べる。実験は Consumer と Producer が 1 対 1 で繋がれている単純なトポロジーにおいて行う。実験トポロジーを図 5.2 に示す。

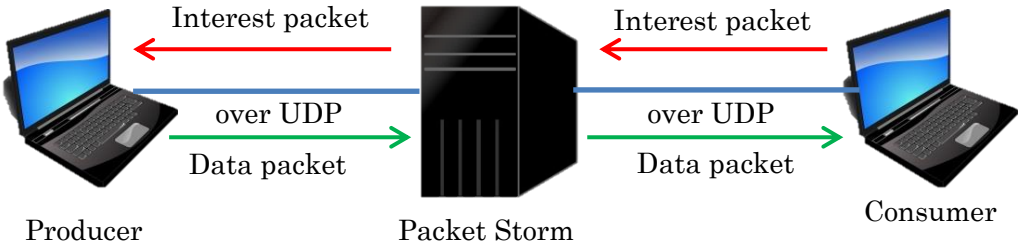


図 5.2 NDN の輻輳制御方式評価実験トポロジー

Consumer と Producer の間には Packet Storm を設置し、複数のネットワーク環境をエミュレートしている。ここで、Packet Storm とは、遅延やパケットロス等、様々な IP ネットワーク環境をエミュレートする機器である。Packet Storm は IP ネットワーク環境をエミュレートする機器であるが、本実験ではピュアな NDN 環境を用意することが不可能であるため、IP 上において NDN プロトコルを用いて通信を行っているため、利用可能である。Producer と Consumer の両者は共に Ubuntu 12.04 LTS が動作しており、NDNx ver 0.3. がインストールいる。また、それぞれ Packet Storm に有線接続されている。NDN プロトコルにおける輻輳制御は NDNx を用いて行うため、NDN の上のプロトコルである IP 層では UDP を用いている。

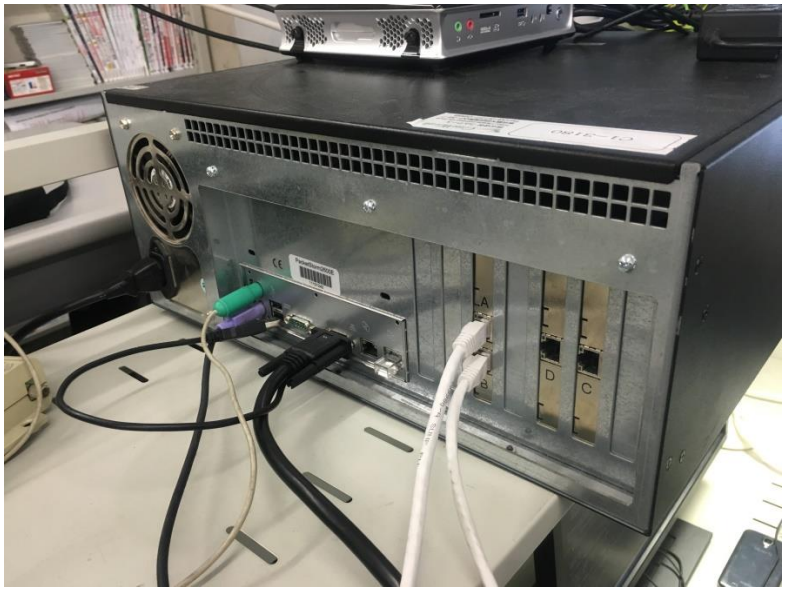


図 5.3 Packet Storm IP network emulator

Packet Storm の設定を 4 通りとし、4 種類の環境を模擬し実験する。

A) Packet Storm の設定なし(ボトルネック無し)

B) 40ms の遅延が発生

C) 10ms~20ms の遅延がガウシアン分布に従い発生

D) 0ms~20ms の遅延がガウシアン分布に従い発生

また、以下に bigbuckbunny1080p.mp4 というコンテンツを転送する場合を例とした Producer と Consumer で入力するコマンドを記載する。

Producer

```
$ ndnd-start   //ndn デーモンの起動
$ ndnsendchunks -b 1024 ndn:/bigbuckbunny1080p.mp4 <
bigbuckbunny1080p.mp4
//転送するコンテンツの名前登録と Data パケットのサイズ決定
```

Consumer

```
$ ndnd-start   //ndn デーモンの起動
$ ndncatchunks2 ndn:/bigbuckbunny1080p.mp4 > bigbuckbunny1080p.mp4
//受信するコンテンツ名を決定し、受信開始
```

ここで、ndnsendchunks とは ndncatchunks2 と対をなすコマンドであり、コンテンツをパブリッシュするコマンドである。-b オプションにより Data パケットのサイズを指定することができ、指定しない場合はデフォルトで 4096 Byte となる。今回は MTU のサイズを超えないようにするため 1024 Byte としている。

5.3.2. 実験結果

5.3.1 で述べた実験環境において、それぞれの輻輳制御方式のそれぞれの環境における結果を示す。また、今回は比較対象として NDNx ver0.3.にデフォルトで実装されている輻輳制御方式を”Default”として用いている。

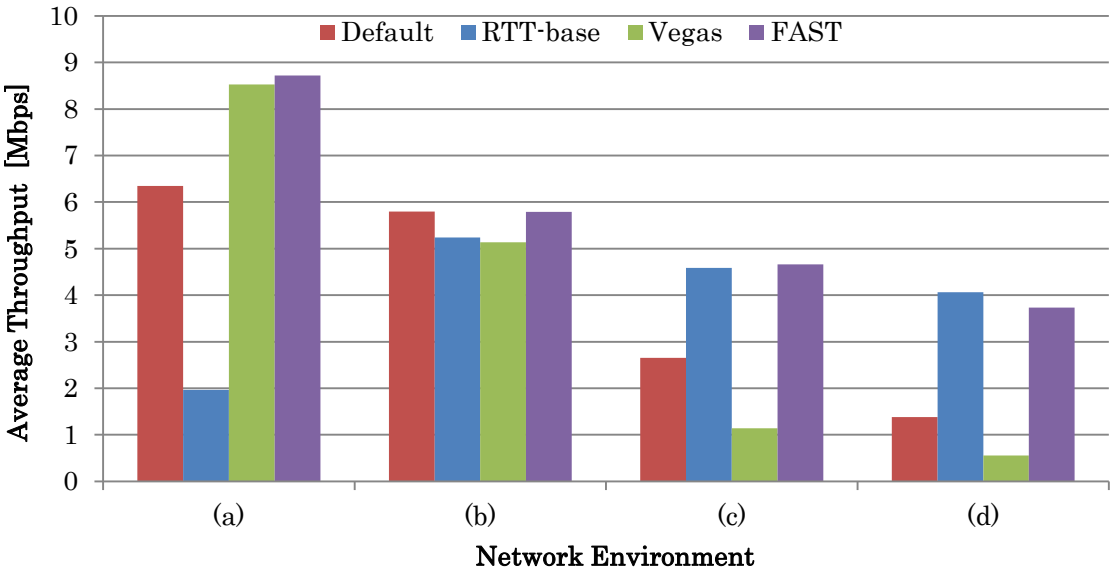


図 5.4 各環境下での各輻輳制御方式の平均スループット

図 5.4 より、経路中にボトルネックが存在しない環境(a)の場合において、デフォルトの方式よりも Vegas 方式、FAST 方式において高いスループットを達成していることが分かる。RTT-base 方式には観測された RTT と一つ前の RTT を比較してウィンドウ制御を行うため、RTT に差が無くウィンドウサイズが上がらないため低いスループットとなっている。40ms の一定遅延が発生する環境(b)については、FAST 方式が Default 方式よりも高いスループットを示しているが、その他はやや低いスループットを表している。10~20ms の遅延がガウシアン分布に従って発生する環境(c)においては、RTT-base 方式と FAST 方式は Default 方式より高いスループットを達成しており、両者は遅延の揺らぎに強いといえる。一方 Vegas 方式については Default 方式より低いスループットを表しており、遅延の揺らぎに弱いといえる。最後に、0~20ms の遅延がガウシアン分布に従って発生する環境(d)について、環境(c)と同様に RTT-base 方式と FAST 方式は Default 方式より高いスループットを達成しており、Default 方式は環境(c)に対して 1/2 ほどに減少しているが、両者はの減少率は小さく、遅延の揺らぎに強いといえる。

次に、上述の実験での環境(d)における Consumer 端末の消費電力量を示す。消費電力量は Watts UP? PRO[15]を使用して計測を行った。

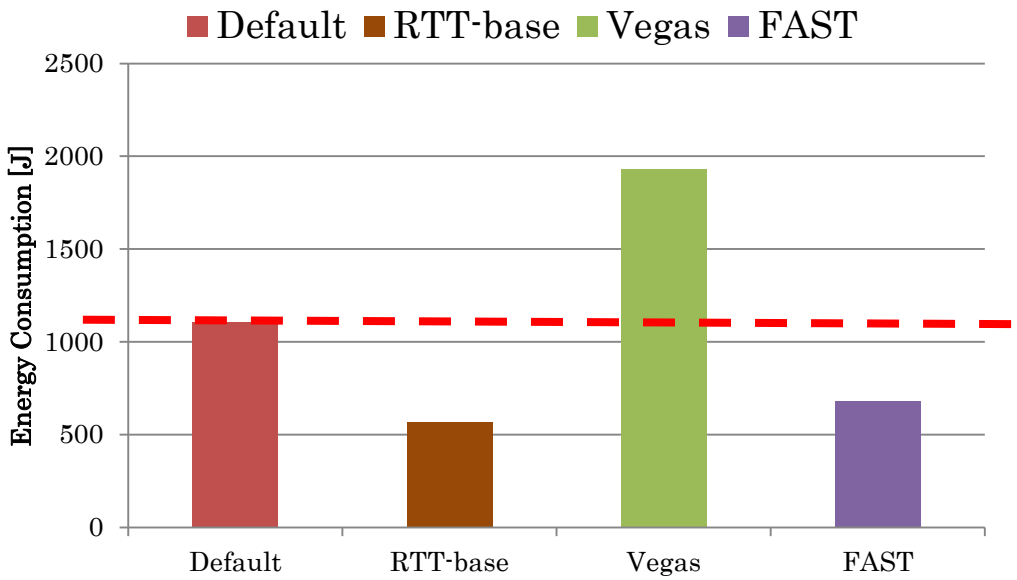


図 5.5 各輻輳制御採用時の環境(d)における Consumer 側端末消費電力量

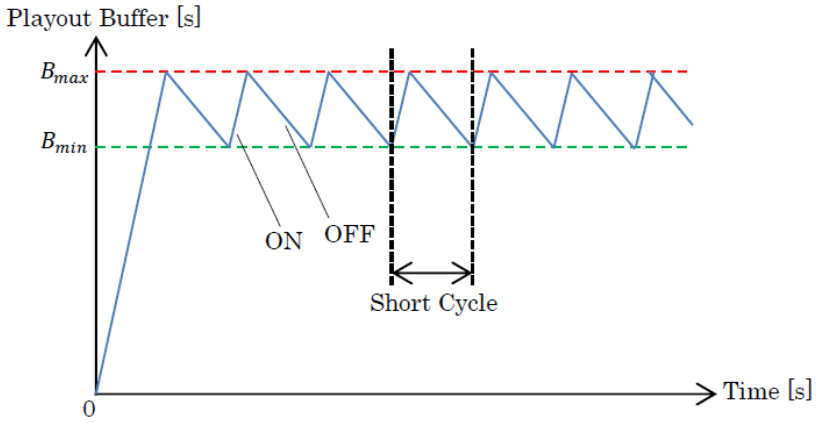
RTT-base 方式、FAST 方式の場合に Default 方式の場合と比較して省電力効果が得られた。これは通信時の端末消費電力量は、端末消費電力([W])と通信時間([s])の積によって求められるため、スループットが高くなると通信時間が短くなるためである。

第 6 章 Interest アグリゲーションと再生バッファを利用した NDN における省電力ビデオストリーミング

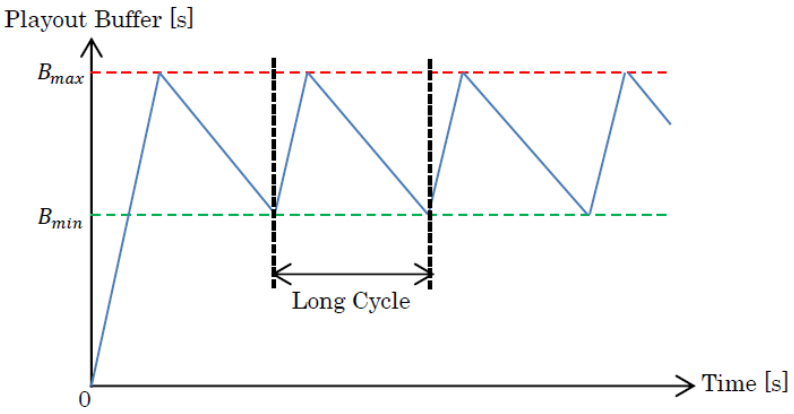
本章では、NDN における省電力ビデオストリーミングを提案する。この提案では Interest アグリゲーションによるスループットの向上と再生バッファ制御によるオーバーヘッド電力の二点に焦点を当てている。

6.1. ビデオストリーミングにおける再生バッファ制御

ビデオストリーミング時の再生バッファ制御に関し、様々な研究がなされている。[16]では、YouTube や Netflix のトラフィックをキャプチャ、解析し、バッファリングメカニズムを”no ON-OFF Cycle”、”short ON-OFF Cycle”、”long ON-OFF Cycle”という 3 つのタイプへと分類している。”ON”フェーズとは、ビデオセグメントがサーバからダウンロードされてバッファリングされている状態を指し、”OFF”フェーズとは、セグメントのデコード、再生のみを行い、通信を行っていない状態を指す。”no ON-OFF Cycle”は、再生前バッファリング時にコンテンツ全体をダウンロードしてから再生を行う、古典的なバッファリング戦略である。”short ON-OFF Cycle”は、短い時間で少量のデータの受信、デコードと再生を周期的に行う。つまり、ON フェーズと OFF フェーズの切り替えの頻度が高いといえ得る。”long ON-OFF Cycle”は、大量のデータの受信、デコードと再生を周期的に行い、かつ周期が長い。つまり、ON フェーズと OFF フェーズの切り替えの頻度が”short ON-OFF Cycle”に比べて低いといえる。図 6.1 にそれぞれのバッファリング戦略におけるバッファの挙動の一例を示す。



(a) “short ON-OFF Cycle”の一例

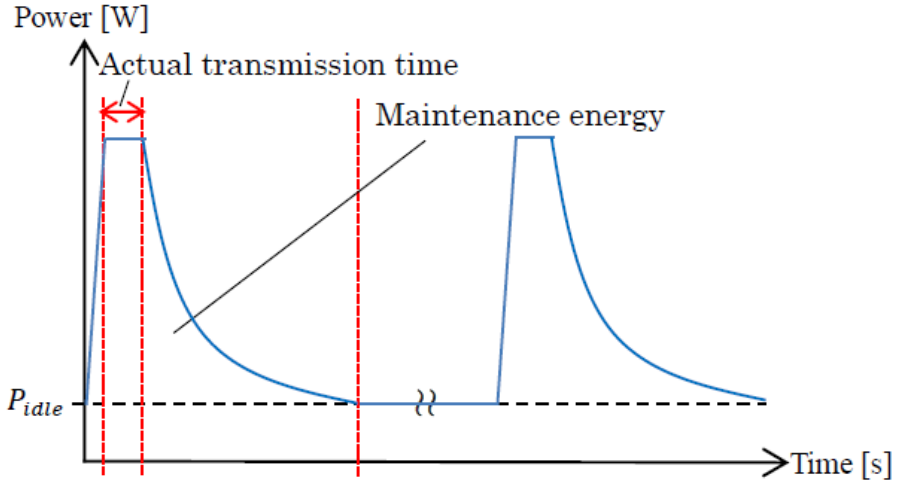


(b) “long ON-OFF Cycle”の一例

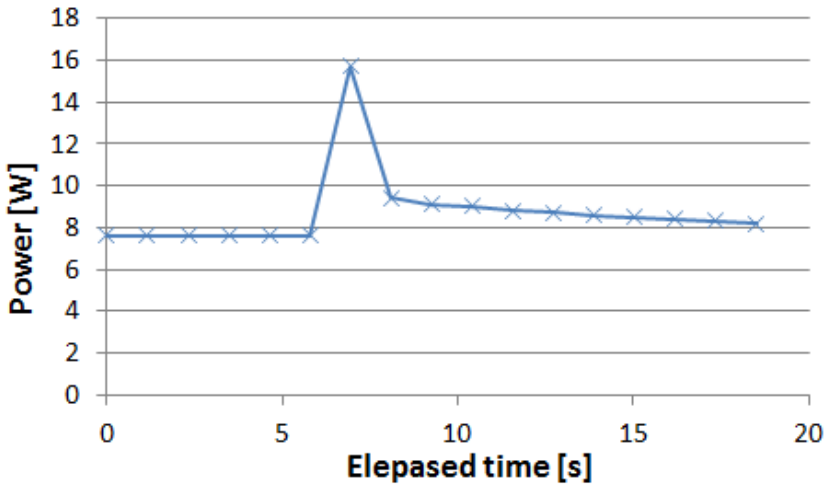
図 6.1 各バッファリング戦略のバッファの挙動の一例

6.2. Wi-Fi 接続時の端末消費電力

[17]では、端末の Wi-Fi 通信時には”maintenance energy”と呼ばれるオーバーヘッドエネルギーが存在すると報告されている。Maintenance energy の一例を図 6.2 に示す。



(a) Maintenance energy の一例



(b) 実際に観測された maintenance energy

図 6.2 maintenance energy

ただし、P_{idle}は待機電力を表す。図のように、通信が始まると端末の消費電力は急激に増加し、通信中は高い消費電力を示す。一方通信終了後は急激に待機電力へ戻らず、緩やかに待機電力へと戻っていく。これを maintenance energy と呼び、毎通信毎に現れるため、待機状態から通信状態への遷移頻度が高いほど余分な電力を消費するといえる。

6.3. NDN における効率的動画配信

Interestアグリゲーションによる NDNにおける効率的ビデオコンテンツ配信について述べる。

6.3.1. Method

端末の通信時消費電力量はスループットに強く依存するため、省電力な動画配信を実現する上で効率的な配信は必要不可欠である。従来の NDN アプリケーションには、4.1 にて述べた TCP Tahoe のようなレガシーな輻輳制御方式が実装されており、利用可能なネットワーク帯域を十分に活用することができていないという問題が存在していた。本章では、Interest アグリゲーションによる効率的動画配信を目的としたスループットの向上につて述べる。本提案では大きく分けて 3 つの要素を含んでいる。

一つ目は、Interest パケットを TCP 上で送信することにより、TCP の再送メカニズムによりアプリケーションレベルでパケットロスを考慮する必要がなくなる。

二つ目は、TCP の輻輳制御を最大限に活用するため、Interest パケットは一つのバーストトラフィックにアグリゲートされ、送信される。従来の NDN アプリケーションでは順次送信されるため、TCP 上で Interest パケットを送信したとしても、上位層の TCP の輻輳制御を十分に活用することができていなかった。

最後に、余分な Interest パケットの送信を解決するため、Consumer に所望するコンテンツのサイズ情報を持たせる。NDN ではすべてのコンテンツは一定サイズのチャンクに分割され(デフォルトで 4096KB)、それぞれに唯一無二の name を付与される。つまり、Consumer は所望のコンテンツを取得するためには複数の Interest パケットを送信しなければならない。つまり、Consumer は所望のコンテンツを取得するのに必要以上の Interest パケットを送信する必要が存在した。この問題を解決するため、本提案ではビデオストリーミング、特に MPEG-DASH を利用したビデオストリーミングでは MPD ファイルへ各ビデオセグメントのサイズ情報を加える拡張を行う。DASH クライアントは初めに MPD ファイルを取得しなければならないため、この拡張された MPD ファイルを参照することにより、式(6.1)を用い、各セグメントを取得するために必要な Interest パケットの数を知ることができる。

$$I = {}^{S_{content}}/_{S_{chunk}} \tag{6.1}$$

ここで、I、$S_{content}$、S_{chunk}、はそれぞれ、所望のコンテンツの取得に必要な Interest パケットの数、所望のコンテンツのサイズ、チャンクサイズを表している。

6.3.2.実装と実験環境

6.3.1 にて述べた手法の評価実験を行うための実装について述べる。Consumer には VM 上に Ubuntu 12.04 LTS をインストールし、Node.js、NDN-JS をインストールし、Producer には NDNx、NDN-TLV をインストールし、NDN ルータの役割を持たせる。ここで、Consumer 側の実装についてさらに詳しく述べる。Consumer 側は NDN-JS を用いて Interest パケットを送信するが、コンテンツを受信するために必要な Interest パケットすべてに name を付与しなければならない。また、name の付与、送信をプログラム中で一つのループで行い、そのループ回数を必要となる Interest パケットの数分とすると、CPU にパケットを送信する余裕がなくなり、結果としてスループットが低くなってしまう。そこで、name の付与と Interest パケットの送信を 100 回行うごとに 1ms のスリープを挿入することにより、Interest パケットのバースト送信を実現している。以下にアルゴリズムを示す。

	Algorithm of Interest Aggregation on Programming
1	FOR (var i = 0; i<100; i++) {
2	send Interest packets;　　//Interest パケットの生成、name の付与を行う関数
3	IF (snd_nxt == content_Interest)
4	break;
5	Sleep 1ms　　//sleep 1ms }

ここで、snd_nxt と content_Interest はそれぞれ、送信した Interest パケットの総数、

conetnt_Interest はコンテンツの受信に必要な Interest パケットの総数を表す。

次に、図 6.3 に実験トポロジーを示す。Consumer と Producer は一対一に有線接続され、Producer には 3 つのビデオコンテンツが格納されている。ビデオコンテンツのサイズはそれぞれ 11.4MB、44.5MB、231.6MB である。

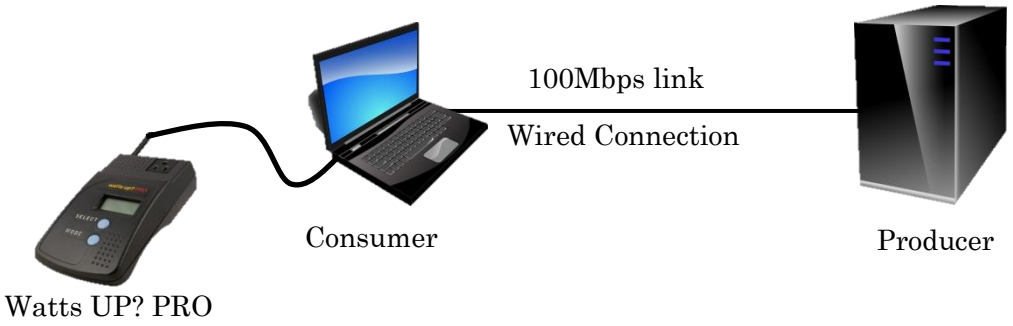


図 6.3 Interest アグリゲーションの効果確認のための実験トポロジー

また、省電力効果を確認するため、消費電力計測ツール Watts UP? PRO を Consumer へ接続し、実験を行う。表 6.1 に Consumer と Producer 端末のスペックを示す。試行回数は 3 回とし、平均を結果として採用する。

表 6.1 各デバイスのスペック

	Consumer	Producer
CPU	Intel Core i5-4310U @ 2.00GHZ	Intel Core i5-3470 @ 3.20GHZ
OS	Ubuntu 12.04 LTS	Ubuntu 14.04 LTS
RAM	2.0GB	8.0GB

6.3.3. 実験結果

図 6.4 に従来の NDN アプリケーションと本提案を実装したアプリケーションのスループットの比較結果を、また図 6.5 に Consumer 端末の通信消費電力量の比較結果を示す。図 6.4 から、本提案は全ての動画コンテンツを受信した場合に従来のアプリケーションよりも高いスループットを達成していることが分かる。また、図 6.5 から、本提案は省電力効果を達成していることが分かる。これは、通信時消費電力量は通信時間に強く依存しており、スループットが高くなるほど通信時間は短くなるためである。以上より、Interest アグリゲーションによる Consumer 端末の省電力効果を確認した。

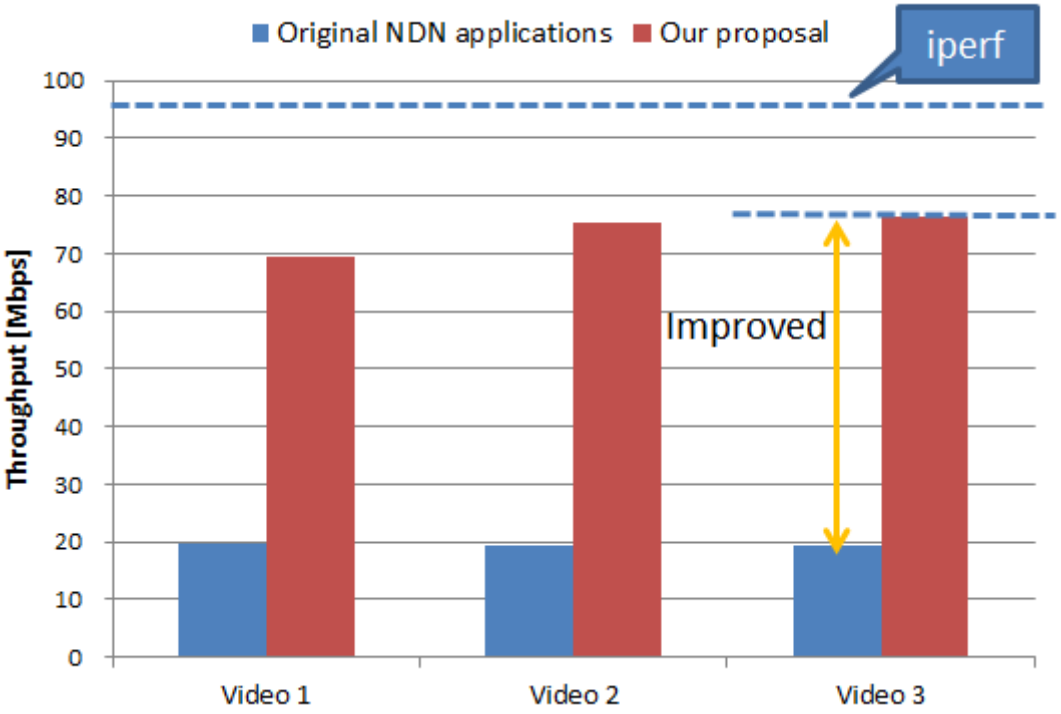


図 6.4 従来の NDN アプリケーションと本提案のスループット比較

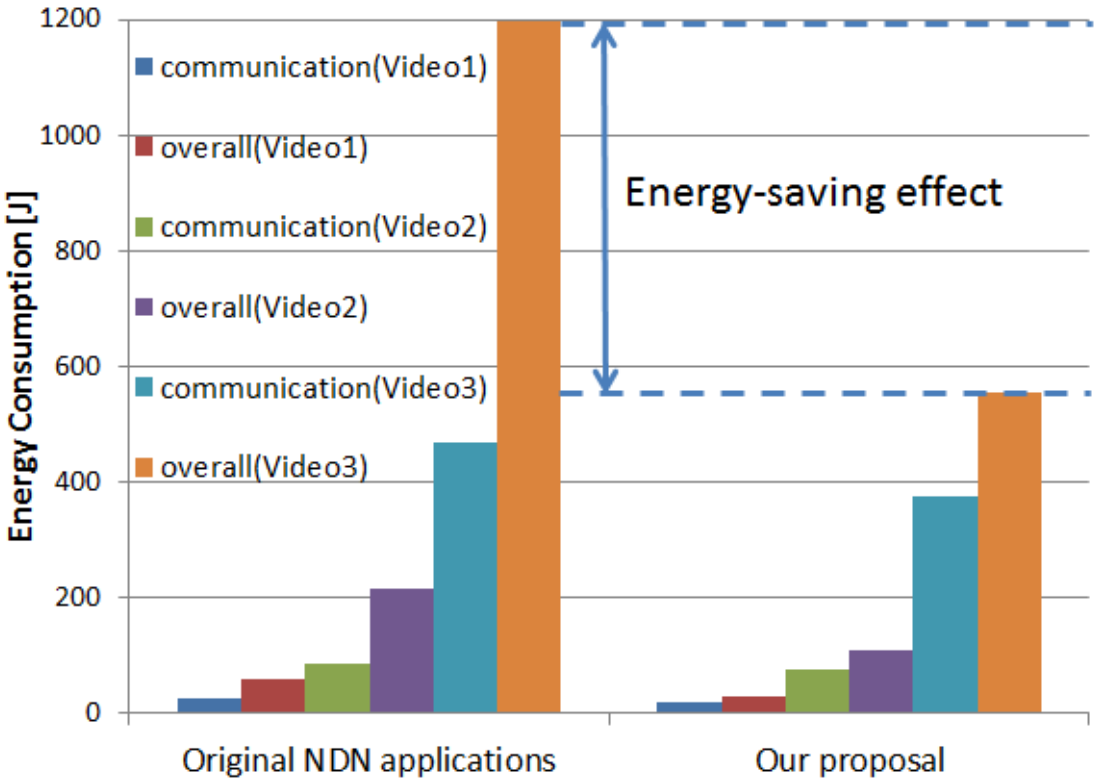


図 6.5 従来の NDN アプリケーションと本提案の通信消費電力量比較

6.4. 再生バッファ制御による省電力ビデオストリーミングの検討

無線通信ビデオストリーミング時の端末消費電力量の削減を達成するため、ストリーミングクライアントサイドの省電力バッファリング戦略について検討する。初めに、本検討で想定するネットワークシステムを図 6.6 に示す。

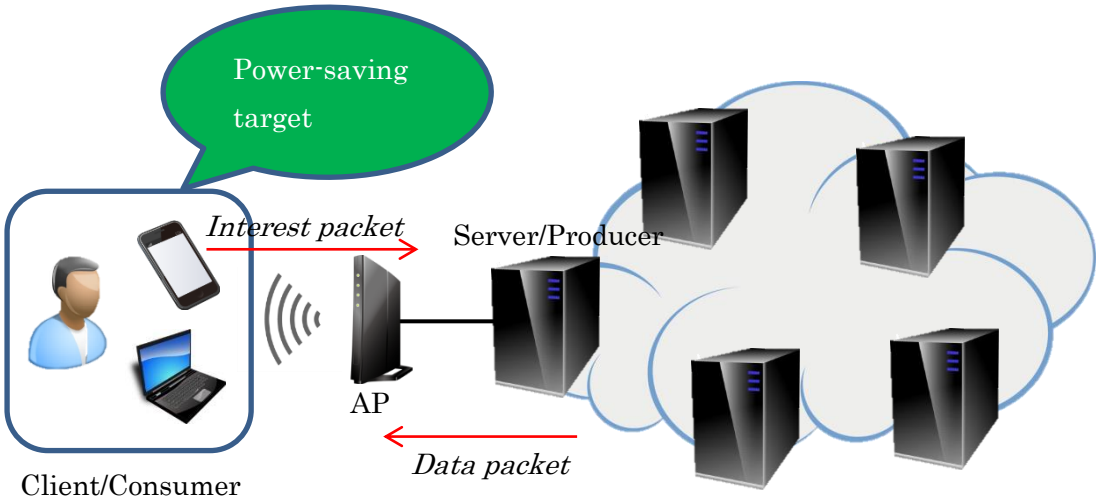


図 6.6 本提案で想定するネットワークシステム

本システムでは、NDN のキャッシングメカニズムにより要求するビデオコンテンツはエッジルーターへ格納されていると仮定し、クライアントはアグリゲートされた Interest パケットを用いてコンテンツを要求する。

6.4.1. Method

6.1 において述べた毎通信後に表れるオーバーヘッドエネルギーの削減を行うことによる省電力効果の検証を行う。ビデオストリーミングにおいてクライアントはフリーズすることのないスムーズな動画再生を実現するため、複数の動画セグメントをバッファリングする必要がある。6.1 で述べた通り、long ON-OFF Cycle のように再生バッファを大きくとると、クライアントは長時間通信を行わずに再生のみを行うことができるため、端末の通信部分は長い待機状態を得ることができる。バッファを大きくとるケースでは図 6.7 に示すようにビデオストリーミング中の端末消費電力は待機電力(P_{idle})まで減少する。一方、short ON-OFF Cycle のように再生バッファが小さい場合、クライアントは数秒しか待機状態へ移行することできず、場合によっては図 6.7 のように待機電力(P_{idle})へ戻る前に次の通信が始まる。以上より、ビデオストリーミングにおける端末消費電力量を検討する上で、再生バッファ制御は重要な要素となっている。本検討では、long ON-OFF Cycle を採用することによる消費電力量削減効果を検証する。また、図 6.7 の待機状態へ移行することによる省電力効果が領域①に、大容量データを一度に受信することによる過剰エネルギーが領域②に示されている。この二つの領域面積にはトレードオフの関係を持ち、領域①が領域②よ

り大きい場合に省電力効果が得られ、領域①が領域②より小さい場合は省電力効果を得ることができないことを表している。

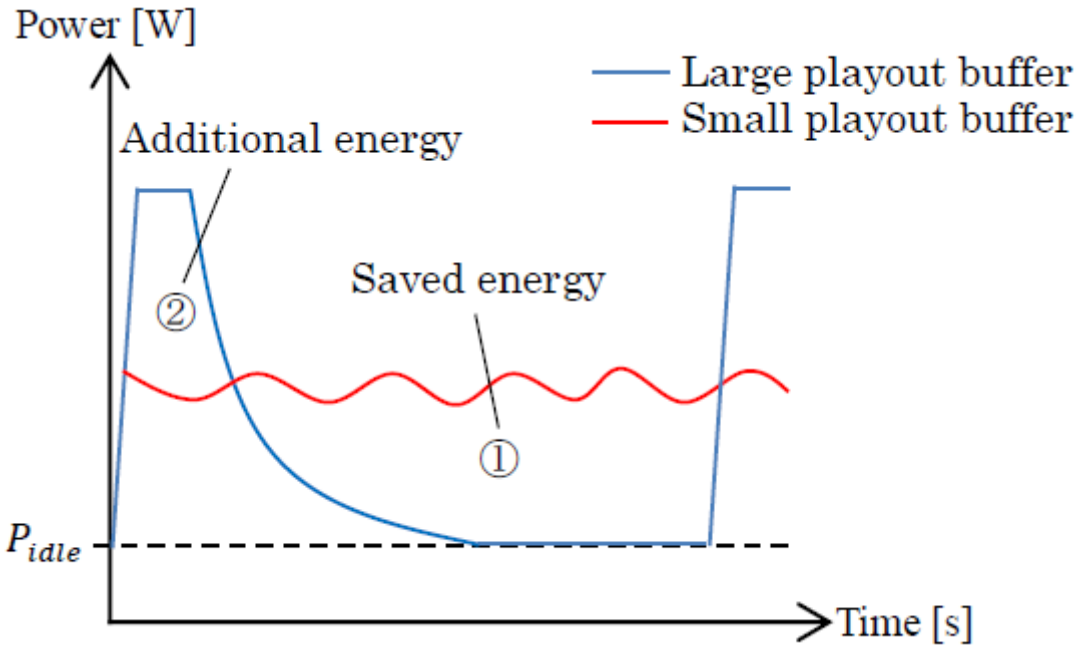


図 6.7 サイズの異なる再生バッファ採用時の端末消費電力のイメージ

6.4.2.実験環境

実機環境において、再生バッファ制御による省電力効果の確認を行う。ビデオセグメントのダウンロードを NDN プロトコルで行うため、Consumer と Producer 端末に NDNx と Node.js をインストールする。実験環境を図 6.8 に示す。本評価実験では、Producer は AP(Access Point)へ有線接続されており、Consumer は AP へ IEEE 802.11ac により無線接続されている。AP は Cisco Aeronet 3700i を、Consumer 端末にはノート PC を利用している。動画コンテンツは”Tears of steel[18]”と呼ばれるビデオクリップを使用し、ストリーミングプロトコルに MPEG-DASH を用いるため、DASHEncoder を用いて H.264/AVC エンコーディング、DASH セグメント化を行い、Producer のリポジトリへ登録されている。この DASH セグメント長は 2 秒としている。また、エンコーディングターゲットビットレートは 3Mbps、7Mbps、20Mbps としているが、本評価実験では再生バッファ制御による純粋な省電力効果を検証するため、配信ビットレートは一定としている。表 6.2 に表されるように、再生バッファ制御を実現するため、ストリーミングクライアントの最小バッファサイズは 20[s]とし、最大バッファサイズを 22[s]、80[s]、140[s]としている。これにより待機状態が 2[s]、60[s]、120[s]となることを表している。また、Consumer 端末の消費電力量を測定するため、消費電力計測ツール Watts UP? PRO を利用しており、通信部分の消費

電力量にのみ着目するため、通信時消費電力から待機消費電力を減算する。

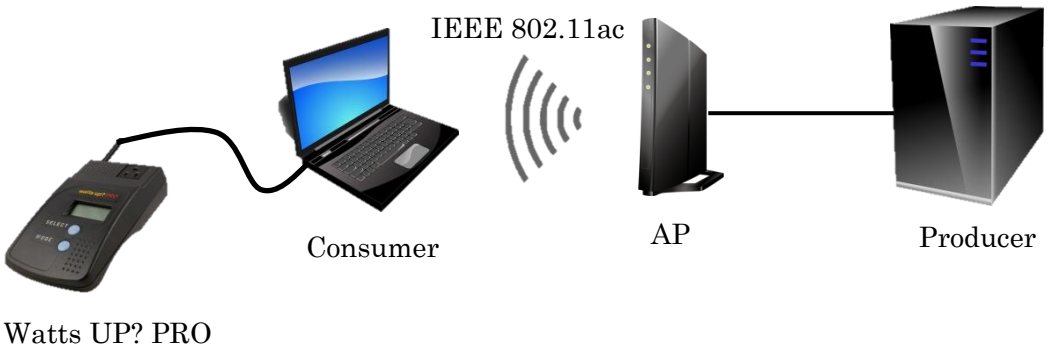


図 6.8 実験環境

表 6.2 再生バッファ制御パラメータ

Playout buffer strategy	Minimum buffer size	Maximum buffer size
Short ON-OFF Cycle	20 [s]	22 [s]
Long ON-OFF Cycle	20 [s]	80 [s]
Long ON-OFF Cycle	20 [s]	140 [s]

6.4.3. 実験結果

図 6.9 に実験結果を示す。ほとんどのケースにおいて再生バッファが大きくなるほど消費電力量が減少していることが分かる。ビデオビットレートが 20 [Mbps]の場合においてのみ、再生バッファサイズが 140 [s]の場合の消費電力量が 80 [s]の場合より大きくなっている。これはバッファサイズを大きくとることによりクライアント端末のメモリのデータ量が増加したことによると考えられる。

本評価実験から、long ON-OFF Cycle(140 [s])の場合、short ON-OFF Cycle の場合と比較して最大で 37%の省電力効果が得られることを確認した。また、再生バッファ制御による省電力効果はビデオコンテンツのビットレートに関係無く観測することができることを確認した。

第 7 章 ビデオ配信における再生バッファ制御に基づく電力モデル提案と省電力化への検討

本章では、再生バッファ制御に基づく NDN 上でのビデオストリーミング時のクライアント端末の消費電力量モデルの提案を行う。

7.1. スマートフォンの LTE 状態遷移

LTE には図 7.1 に表すように RRC_CNNECTED と RRX_IDLE と呼ばれる二つの状態が存在する。また、RRC_CONNECTED では UE(User Experience)はさらに Continuous Reception、Short DRX、Long DRX という三つの状態が存在し、RRC_IDLE には DRX という一つの状態のみ存在する。初めに UE が RRC_IDLE にいるときにパケットを受信/送信すると、パケットサイズの大きさに関わらず RRC_IDLE から RRC_CONNECTED へ数秒の遅延後に移行する。RRC_CONNECTED へ移行後、UE は Continuous Reception 状態へと移行し、制御メッセージである PDCCH(Physical Downlink Control Channel)のモニタリングを開始する。また、UE は inactivity timer($\triangle t_i$)と呼ばれる時間のカウントを始め、$\triangle t_i$ はパケットの送信/受信がある度にリセットされる。パケットの送受信が無く $\triangle t_i$ が経過すると UE は Short DRX へと移行する。LTE では高い QoS を保ったまま省電力にコネクションを保つために DRX(Discontinuous Reception)と呼ばれるアルゴリズムが実装されている。DRX の様子を図 7.2 に示す。RRC_CONNECTED と RRC_IDLE の DRX は似通ったメカニズムを持つが、パラメータが異なる。DRX cycle は UE が PDCCH をモニタリングしている間は On Duration を含んでおり、残りの時間は消費電力を抑えるために待機状態となる。バッテリーの消費電力削減と通信のレイテンシのトレードオフは指標となっている。On Duration の時間が固定値である場合、DRX のサイクルを長くするとユーザにとって少量の遅延が発生するが、消費電力量の削減につながり、DRX のサイクルを短くすると消費電力量多くなるがユーザにとって遅延が小さくなる。Short DRX と Long DRX の On Duration の時間は同様であり、上記の対立した要求を満たすために考案された。UE が short DRX へ入ると Short Cycle Timer($\triangle t_{is}$)のカウントが始まり、$\triangle t_{is}$ が経過するとデータの送受信が無ければ Long DRX へと遷移し、送受信がある場合は Continuous Reception へと再び遷移する。また、送受信が始まり UE が Continuous Reception へ遷移すると UE は Tail timer($\triangle t_{tail}$)のカウントを始める。また、このタイマーはパケットの送受信がある度にリセットされる。$\triangle t_{tail}$ が経過すると UE は RRC_CONNECTED から RRC_IDLE へ遷移する。

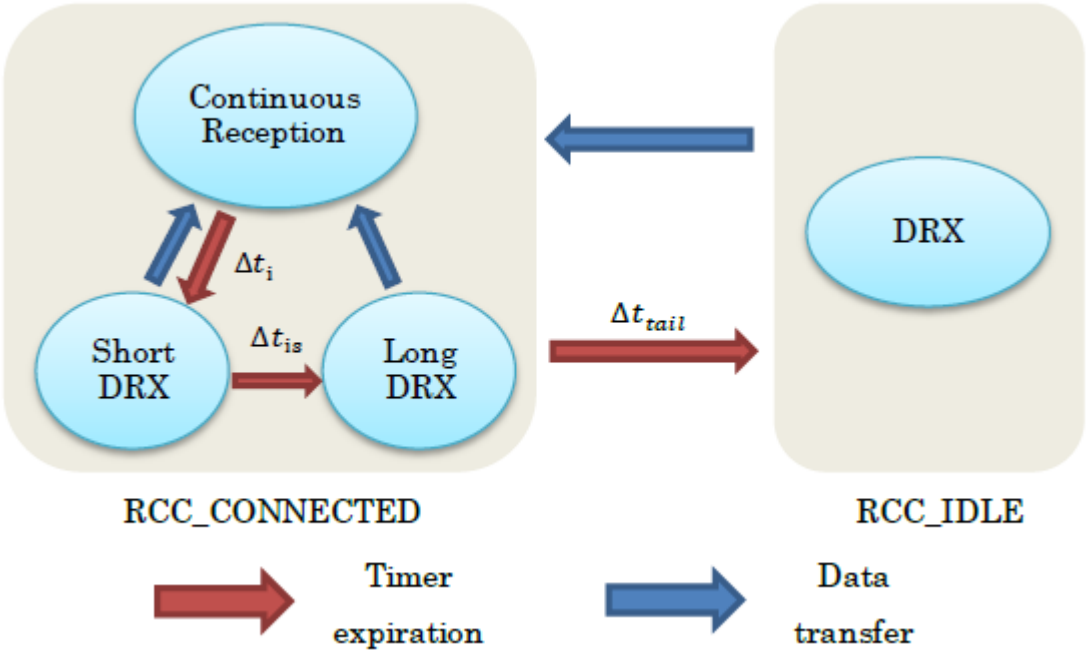


図 7.1 LTE の状態遷移の様子

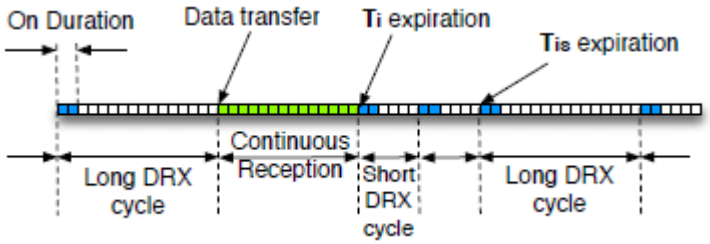


図 7.2 RRC_CONNECTED における DRX のイメージ([17]より)

7.1.1.スマートフォンの LTE 状態遷移と端末消費電力

7.1 で述べた通り、LTE には大きく分けて二つの状態が存在する。また、それぞれの状態では端末依存の消費電力が発生しており、ビデオストリーミング時の消費電力量の削減を検討する上で重要な要素のうちの一つである。図 7.3 に LTE 通信時、通信後の端末消費電力のイメージ、図 7.4 に実際に観測された消費電力量の一例を示す。また、端末は SC-04E Galaxy S4 を使用し、消費電力計測ツールは Monsoon Power Monitor[19]を使用する。

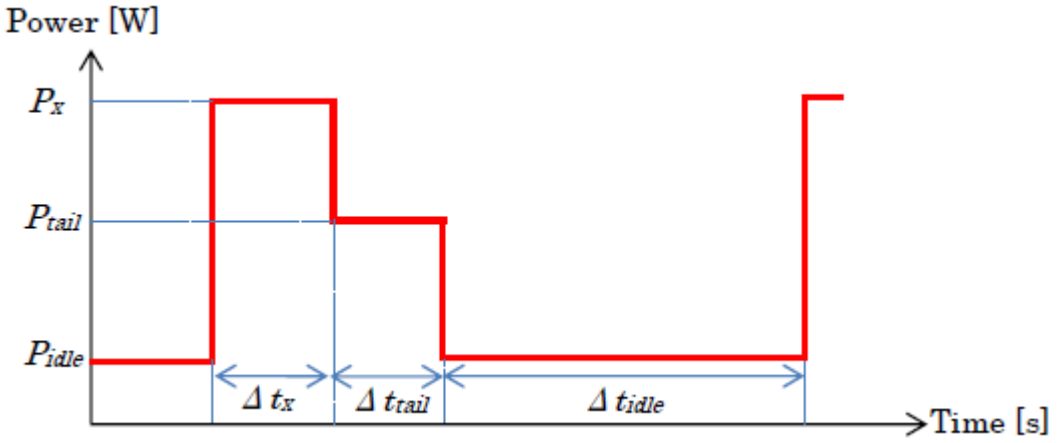


図 7.3 LTE 通信時の端末消費電力のイメージ

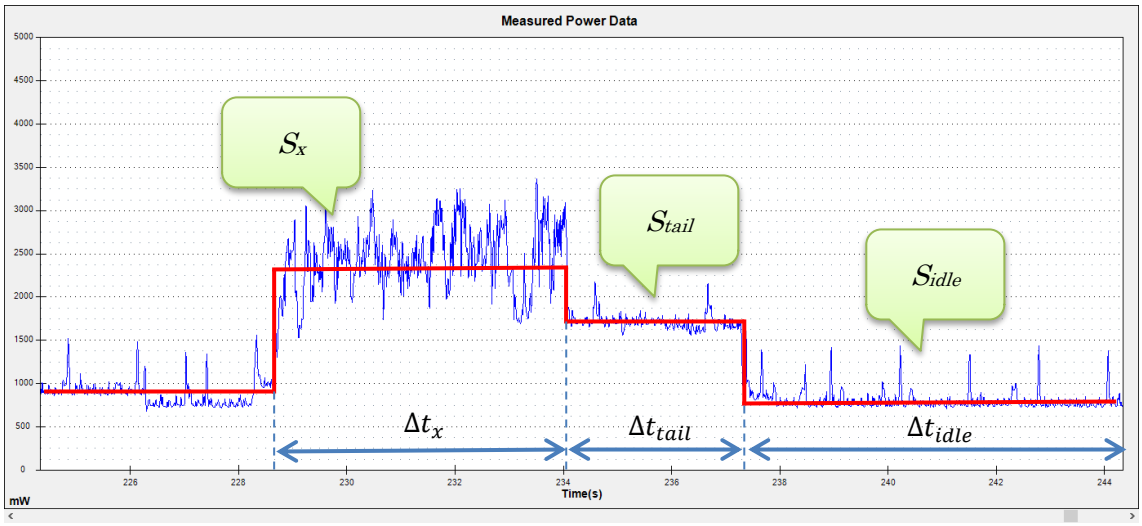


図 7.4 観測された端末消費電力

図 7.3、図 7.4 のように、消費電力の観点から観察すると、通信時、通信後の端末消費電力の状態は大きく分けて、S_x、S_{tail}、S_{idle} の三つに分けることができ、それぞれ通信状態、tail energy 出現状態、待機状態を表す。上二つの図のように通信が始まると端末消費電力は急激に増加し、通信時は高い消費電力を示す。通信終了後は消費電力はやや下がり、おおよそ一定時間通信状態より低く、待機状態より高い電力を示す。最後に、通信終了からΔt_{tail}経過後に待機状態へと戻る。この状態 S_{tail} は毎通信毎に出現するため、余分なエネルギー消費となり得る。また、S_x において示す消費電力量は通信時のスループットに依存する。

7.2. ビデオストリーミング時の端末消費電力のモデル化

7.1.1 にて述べた状態による消費電力の違いを利用し、ビデオストリーミング時のクライ

アント端末の消費電力量のモデル化を行う。本提案では、ビデオストリーミング中の消費電力の一部をサイクルとしてとらえ、1 サイクルについてモデル化する。その様子を図 7.5 に示す。

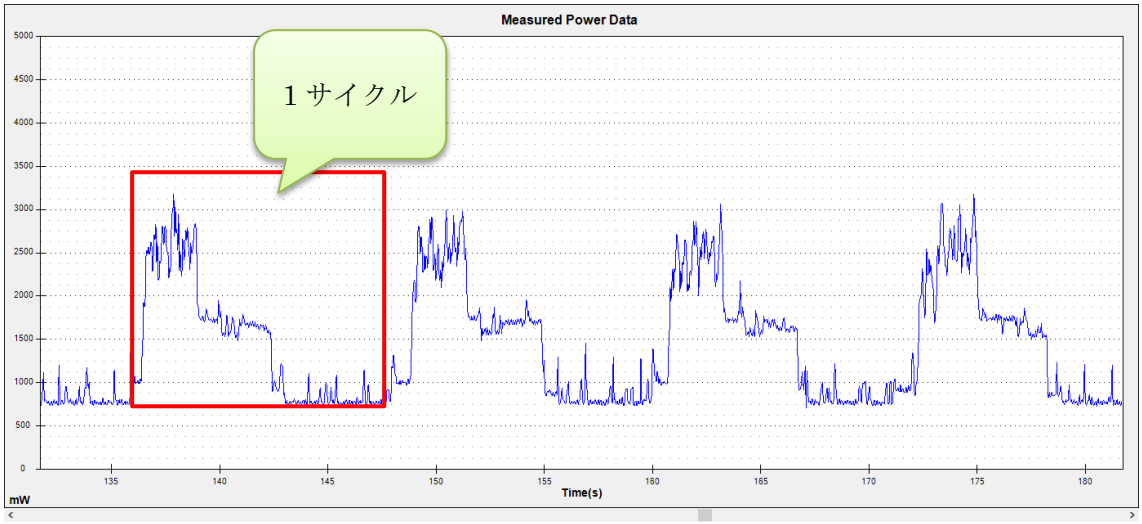


図 7.5 モデル化の対象とするビデオストリーミング中の端末消費電力の 1 サイクル

図 7.5 のように、ビデオストリーミング中のクライアント端末の消費電力は最初と最後を除く場合、S_x、S_{tail}、S_{idle}の繰り返しで構成されており、図のようにそれぞれ一回ずつ含むものを 1 サイクルと定義し、モデルを構築する。1 サイクルに消費される電力量を E_{1cycle} とするとし、S_x、S_{tail}、S_{idle} 各状態における消費電力量をそれぞれ E_x、E_{tail}、E_{idle} とすると、E_{1cycle} を式(7.1)のように表すことができる。

$$E_{1cycle} = E_x + E_{tail} + E_{idle} \tag{7.1}$$

また、3 状態それぞれにおける消費電力をそれぞれ P_x、P_{tail}、P_{idle} とし、それぞれ 3 状態にとどまる時間Δt_x、Δt_{tail}、Δt_{idle} を用いると、

$$E_{1cycle} = \bar{P}_x \cdot \Delta t_x + \bar{P}_{tail} \cdot \Delta t_{tail} + \bar{P}_{idle} \cdot \Delta t_{idle} \tag{7.2}$$

と表すことができる。ここで$\bar{P}_x$、$\bar{P}_{tail}$、$\bar{P}_{idle}$はそれぞれ 3 状態における消費電力の平均を表す。

ここで、それぞれの状態にとどまる時間についてさらに詳しく述べる。まず、Δt_x は通信時間であるため、受信するデータ合計サイズをスループットで除算することで求められる。ここで、クライアントの最大再生バッファを B_{max}、最小再生バッファを B_{min} とし、MPEG-DASH にて選択されたレプレゼンテーションを R、スループットを BW、DASH セグメントのサイズを k [s]とすると、Δt_x は式(7.3)のように表すことができる。

$$\Delta t_x = \begin{cases} \dfrac{(B_{max} - B_{min}) \times R}{BW}, & \Delta t_x < \mathrm{k} \\ \dfrac{(B_{max} - B_{min} + \Delta t_x) \times R}{BW}, & \Delta t_x \geq \mathrm{k} \end{cases} \tag{7.3}$$

ここで、$\Delta t_x \geq \mathrm{k}$とは通信時間が DASH セグメントの時間より長い場合のことを表し、その場合さらに多くのデータを受信する必要があることを表す。式(7.3)をΔt_xについて解くと、

$$\Delta t_x = \begin{cases} \dfrac{(B_{max} - B_{min}) \times R}{BW}, & \Delta t_x < \mathrm{k} \\ \dfrac{(B_{max} - B_{min}) \times R}{BW - R}, & \Delta t_x \geq \mathrm{k} \end{cases} \tag{7.4}$$

と表すことができる。

次に、Δt_{tail}は tail energy の出現時間であり、時間は固定値である[17]。デバイス依存の固定値を Tと置く。また、通信時間Δt_x経過後の 1 サイクルの残り時間は、

$$B_{max} - B_{min} - \Delta t_x + M \tag{7.5}$$

と表すことができる。ここで M は$\Delta t_x > M$を満たす最大の k の倍数を表す。これは、Δt_xが kより大きい場合、つまり通信時間が DASH セグメント長より大きい場合は予定よりも多くのセグメントを取得しなければならないためである。また、この残り時間 $B_{max} - B_{min} - \Delta t_x + M$が Tより大きい場合、1 サイクルの中に待機状態 S_{idle}が存在することができ、Δt_{tail}は Tとなるが、Tより小さい場合、待機状態 S_{idle}へ遷移する前に次のサイクルが始まってしまうため、tail energy の出現時間であるΔt_{tail}は$B_{max} - B_{min} - \Delta t_x + M$となる。以上をまとめると、$B_{max} - B_{min} - \Delta t_x + M$は式(7.6)のように表すことができる。

$$\Delta t_{tail} = \begin{cases} B_{max} - B_{min} - \Delta t_x + M, \\ (B_{max} - B_{min} - \Delta t_x + M) \leq T \\ T, \\ (B_{max} - B_{min} - \Delta t_x + M) > T \end{cases} \tag{7.6}$$

最後にΔt_{idle}について、上で述べたように$B_{max} - B_{min} - \Delta t_x + M$が Tより小さい場合は待機状態 S_{idle}へ遷移する前に次のサイクルが始まってしまうため、Δt_{idle}は 0 となる。一方、$B_{max} - B_{min} - \Delta t_x + M$が T より大きい場合は 1 サイクル全体の時間から通信時間と tail energy 出現時間を引いた$B_{max} - B_{min} - t_{tail} - \Delta t_x + M$となる。以上をまとめると$\Delta t_{idle}$は式(7.7)のように表すことができる。

$$\Delta t_{idle} = \begin{cases} 0, \\ (B_{max} - B_{min} - \Delta t_x + M) \leq T \\ B_{max} - B_{min} - t_{tail} - \Delta t_x + M, \\ (B_{max} - B_{min} - \Delta t_x + M) > T \end{cases} \tag{7.7}$$

また、表 7.1 にそれぞれのパラメータの定義をまとめる。

表 7.1 各パラメータの定義

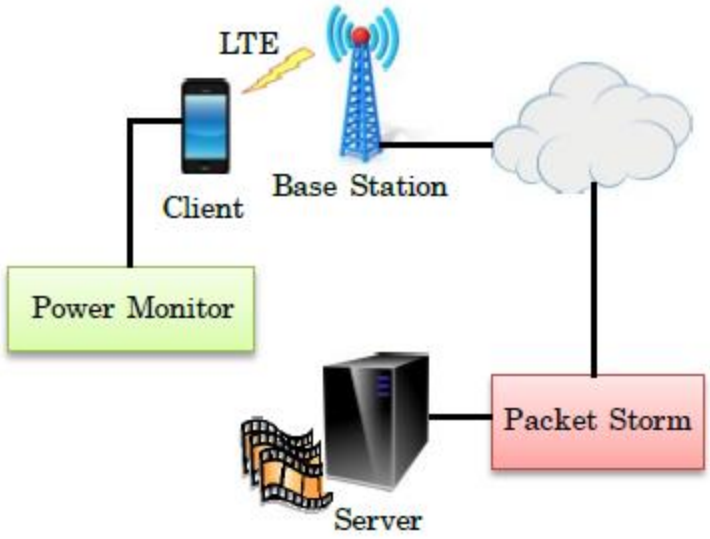
Parameter	Definition
E_{1cycle} [J]	1 サイクルにおける全体消費電力量
E_x [J]	通信時の消費電力量
E_{tail} [J]	Tail energy の消費電力量
E_{idle} [J]	待機消費電力量
$\bar{P}_x$ [W]	通信時の平均消費電力
$\bar{P}_{tail}$ [W]	Tail energy の平均消費電力
$\bar{P}_{idle}$ [W]	平均待機消費電力
Δt_x [s]	通信時間
Δt_{tail} [s]	Tail energy 出現時間
Δt_{idle} [s]	待機時間
B_{max} [s]	最大バッファサイズ
B_{min} [s]	最小バッファサイズ
BW [Mbps]	通信時平均スループット
R [Mbps]	各 DASH セグメントのレプレゼンテーション
k [s]	セグメント長
T [s]	Tail energy の最大出現時間

7.2.1.通信スループットと端末消費電力の関係

通信時の端末消費電力は通信スループットに依存する[17]。スループットと消費電力の関係を求めモデルへ組み込むため、評価実験を行う。実験環境を図 7.6 に示す。実験は NDN と IP の 2 パターンにて実験を行う。NDN におけるパターンでは、NDN-JS を利用し、クライアントはブラウザを利用して NDN プロトコルでコンテンツの受信を行う。通信開始時から消費電力の計測を開始し、NDN プロトコルの上位層で動作している TCP のスロースタートを考慮して初めの 5 秒間は省略する。また、測定データを 1 秒ごとに分割し、1 秒の平均スループットと平均消費電力をデータとして利用する。IP におけるパターンにおいてもクライアントはブラウザで HTML ファイルを読み込み、コンテンツのダウンロードを行い、通信中のスループットと消費電力の関係を結果として利用する。また、IP における実験では、一部のデータにおいて、低スループットにおけるデータを取得するため、クライアントとサーバ間に IP ネットワークエミュレータ Packet Storm を挿入し、帯域を絞ることにより低スループット状態を実現している。実験結果を図 7.7~7.9 に示す。



(a) 帯域を絞らない場合(NDN、IP の一部で使用)



(b) 帯域を絞る場合(IP の一部で使用)

図 7.6 スループット vs 通信消費電力の実験

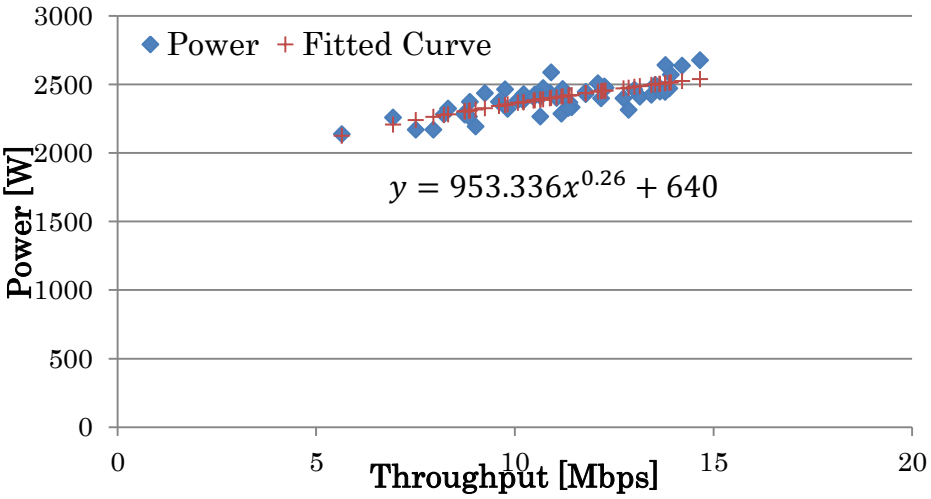


図 7.7 スループット vs 通信時端末消費電力(NDN)

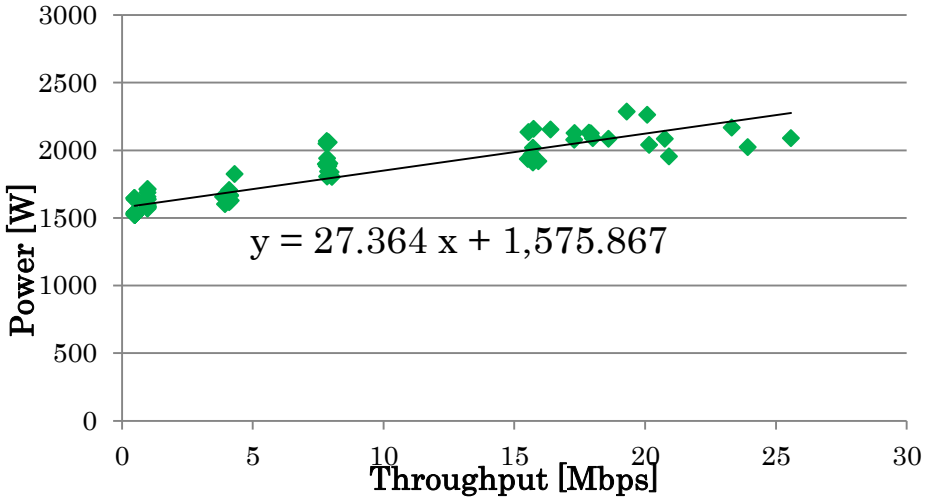


図 7.8 スループット vs 通信時端末消費電力(IP、Packet Storm 有)

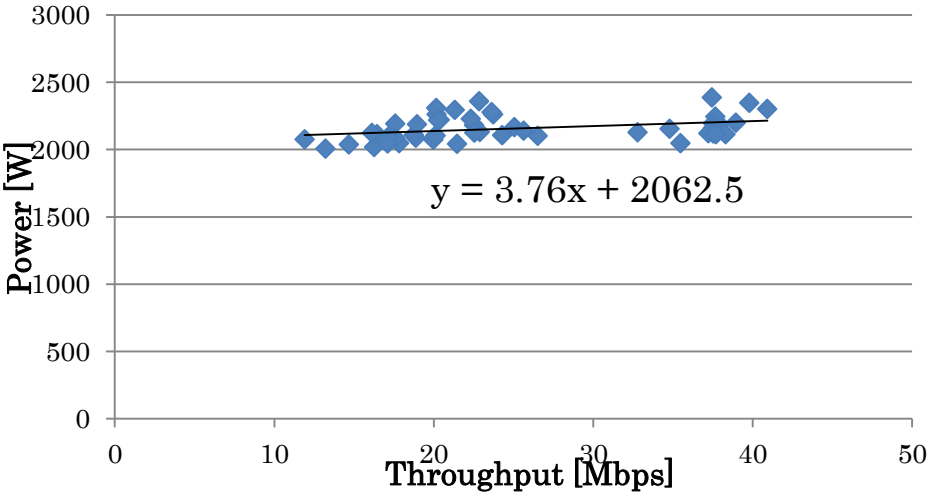


図 7.9 スループット vs 通信時端末消費電力(IP、Packet Storm 無し)

また、図 7.8 と図 7.9 から、スループットが 10~20 [Mbps]あたりではおおよそ同じ消費電力を示しているため、Packet Storm を利用して基地局のバックグランドにボトルネックを発生させてスループットをコントロールする場合も、コントロールしない場合と同様とみなせると仮定すると、図 7.10 のように統合することができる。また、640 という値は、待機電力を表している。

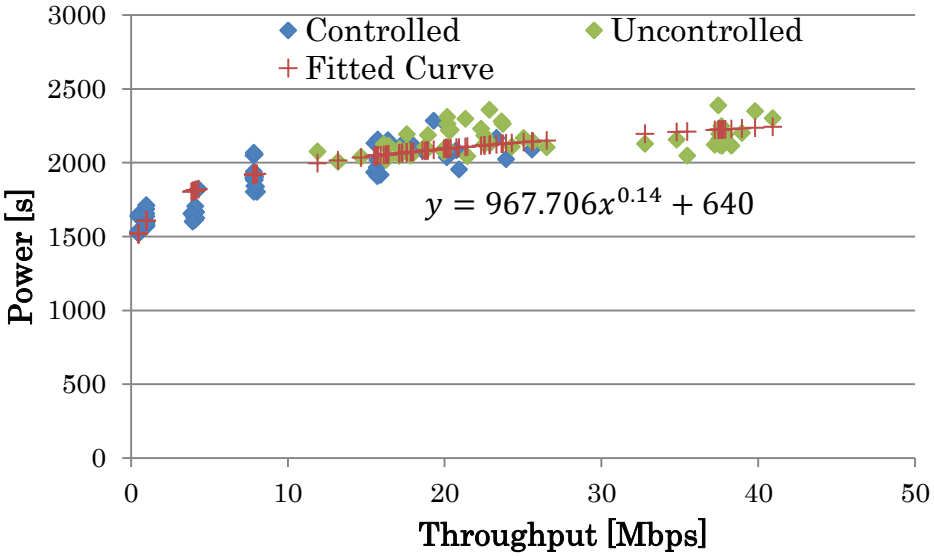


図 7.10 スループット vs 消費電力(IP、統合後)

以上の図 7.7 と図 7.10 の近似式を利用することにより、消費電力量モデルにおける$\bar{P}$を求めることができる。

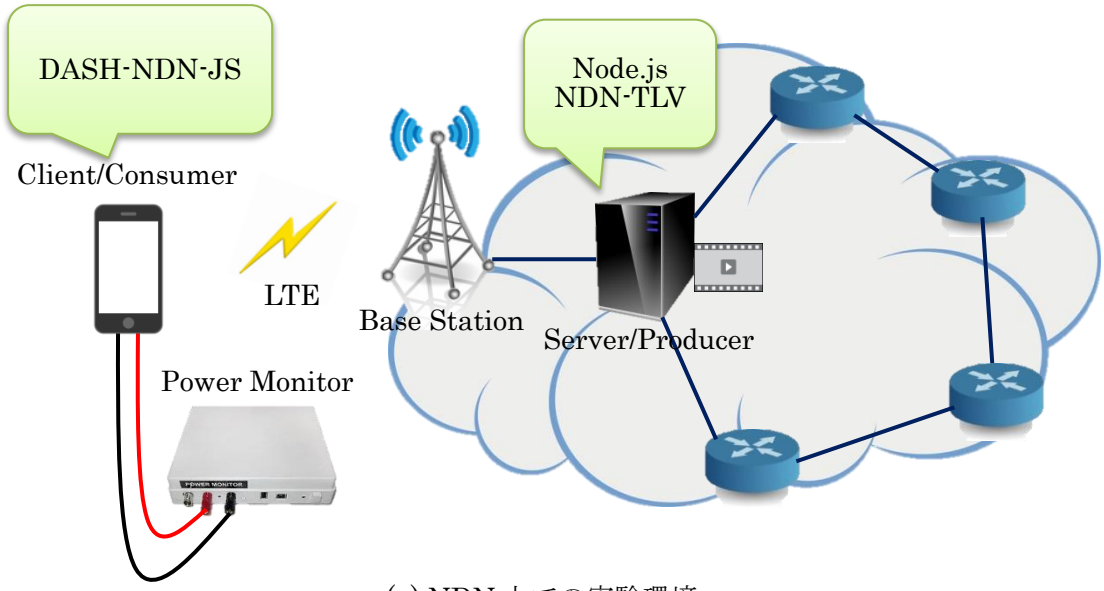
7.3. 消費電力量モデルの検証

提案する消費電力量モデルの検証のため、モデル値と実測値の比較を行う。評価実験は NDN と IP の 2 パターンで行う。

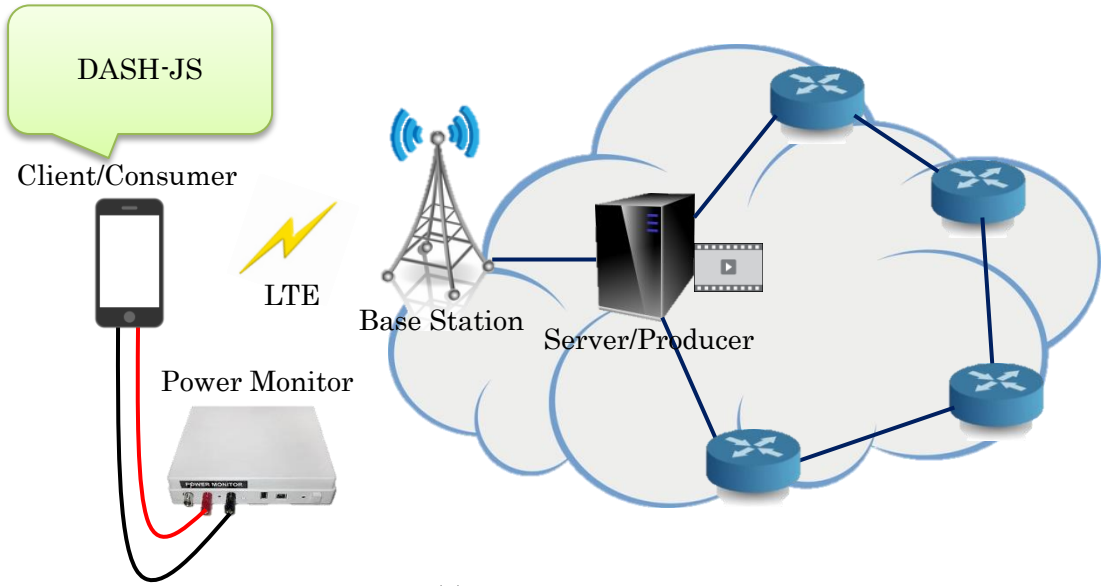
7.3.1. 実験環境

図 7.11 に実験環境を示す。図 7.11-(a)の NDN における実験では、NDN プロトコルを用いてビデオストリーミングを実現するため DASH-NDN-JS[20]を利用し、Server/Producer では WebSocket Proxy と NDN-TLV が動作しており、クライアントは LTE 通信により Server/Producer にある DASH-NDN-JS のソースコードを読みこみ、配信を行う。消費電力の実測には Monsoon Power Monitor を使用する。図 7.11-(b)の IP における実験においては、NDN における実験とアプリケーションが異なり、Client/Consumer は DASH-JS を読み込み DASH 配信を行う。また、両実験において簡略化のため選択レプレゼンテーションは 1 [Mbps]固定とし、通信部分の消費電力に焦点を当てるため、動画のデコード、再生

は行っていない。



(a) NDN 上での実験環境



(b) IP 上での実験

図 7.11 モデル検証評価実験の実験環境

また、モデルの検証を行うにあたり、モデルに必要となる端末依存の実測値とその他モデルのパラメータのまとめを表 7.2 に示す。

表 7.2 検証評価実験におけるモデルパラメータ

(a) NDN 上でのパラメータ

Parameter	Value
$\bar{P}_x$ [W]	$\bar{P}_x = 953.336BW^{0.26} + 640$
$\bar{P}_{tail}$ [W]	1832
$\bar{P}_{idle}$ [W]	900
B_{max} [s]	24, 26, 28, 30, 40, 50, 60
B_{min} [s]	20
R [Mbps]	1Mbps
k [s]	2
T [s]	3.63

(b) IP 上でのパラメータ

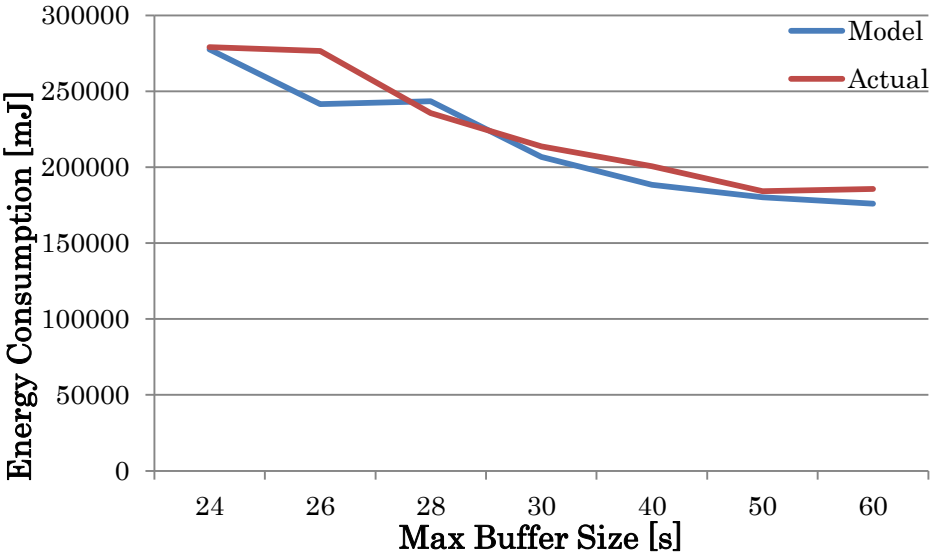
Parameter	Value
$\bar{P}_x$ [W]	$y = 967.706x^{0.14} + 640$
$\bar{P}_{tail}$ [W]	1850
$\bar{P}_{idle}$ [W]	1000
B_{max} [s]	24, 26, 28, 30, 40, 50, 60
B_{min} [s]	20
R [Mbps]	1Mbps
k [s]	2
T [s]	3.63

表 7.2 中の赤字は端末依存の数値となっており、$\bar{P}_{tail}$、$\bar{P}_{idle}$、Tは観測された平均値を利用している。

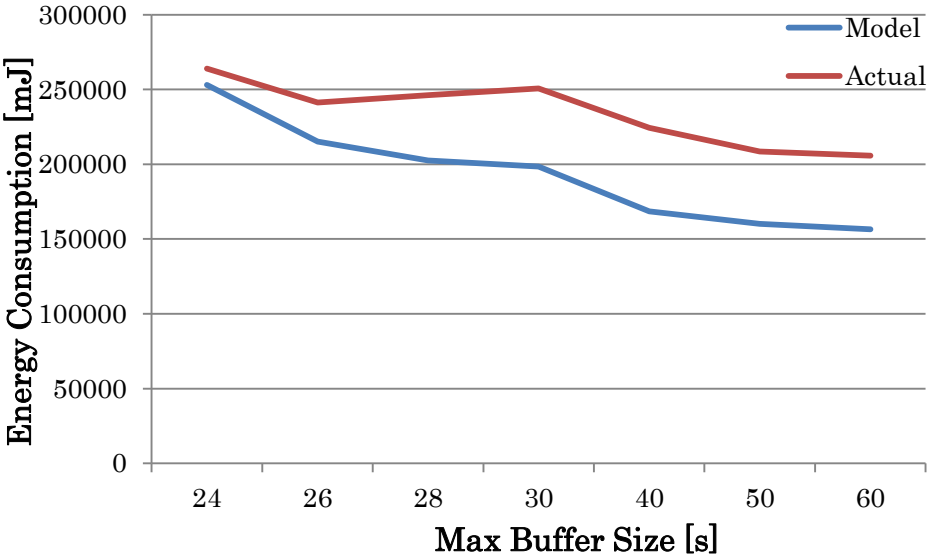
7.3.2. 実験結果および考察

実験結果を図 7.12 に示す。図 7.12 より、最大バッファサイズが大きくなるほど消費電力量は低くなるといえる。つまり、再生バッファのサイズが大きくなるほど省電力となる。これは、再生バッファサイズを大きくとるほど、ビデオストリーミング中の通信頻度が下がり、それに伴い 7.1 で述べた tail energy の出現頻度が下がるため、余分なエネルギー消費が削減されるためである。NDN における検証ではおおよそ傾向が一致しているが、IP における実験ではやや値に差が存在する。これは、IP における実験で利用した DASH-JS の実装によるものである。本実験では、最大で NDN における実験のうちモデル値は 37%、実測値では 34%の省電力効果を確認し、IP における実験のうちモデル値は 38%、実測値では 22%の省電力効果を確認した。また、全体の消費電力量から待機電力量を引くことにより通

信に消費された電力量のみに焦点を当てると、最大で NDN における実験のうちモデル値は 73%、実測値では 65%の省電力効果を確認し、IP における実験のうちモデル値は 83%、実測値では 56%の省電力効果を確認した。



(a) NDN 上における実験結果



(b) IP 上における実験結果

図 7.12 モデルの検証評価実験結果

第 8 章 総括

8.1. まとめ

本論文では、ビデオストリーミング時の端末消費電力のモデル化、再生バッファ制御に基づく消費電力量削減手法の検討を行った。Wi-Fi 接続時のノート PC、LTE 接続時のスマートフォンには共に通信終了後に余分な電力消費の原因となり得る maintenance energy と tail energy がそれぞれ存在した。ビデオストリーミングクライアントにはネットワーク状態に対してロバストなストリーミングを実現するために再生バッファが実装されており、再生バッファの容量を制御することにより、ストリーミング中の端末の通信頻度を変化させることができる。通信頻度が低くなることにより、毎通信後に出現する余分なエネルギーの出現頻度も低くなり、結果としてストリーミング全体で省電力効果が得られる。また、スマートフォンの LTE 通信時消費電力は大きく三つの状態に分けることができるため、その特性を利用してビデオストリーミング時の端末消費電力量のモデル式を構築した。再生バッファサイズを変化させ、省電力効果を確認すると同時にモデル式と実測値の比較を行ったところ、傾向はおおよそ一致し、最大で NDN における実験のうちモデル値は 37%、実測値では 34%の省電力効果を確認し、IP における実験のうちモデル値は 38%、実測値では 22%の省電力効果を確認した。また、全体の消費電力量から待機電力量を引くことにより通信に消費された電力量のみに焦点を当てると、最大で NDN における実験のうちモデル値は 73%、実測値では 65%の省電力効果を確認し、IP における実験のうちモデル値は 83%、実測値では 56%の省電力効果を確認した。

8.2. 今後の展望

本論文にて提案したビデオストリーミング時の端末消費電力量モデルには改善が必要とされる。本モデルでは tail energy の出現時間を固定値としているが、実際には値に振れ幅が存在したため、様々な実験により傾向を見つけ、モデル化することにより、端末消費電力量モデルの精度が向上すると考えられる。また、NDN における実験において、本提案ではすべて受信トラフィックとして扱っているが、実際は Interest パケットの送信が含まれるため、送信トラフィックの消費電力と受信トラフィックの消費電力を分けてモデルに組み込むことにより、コンテンツ受信時消費電力の精度が向上すると考えられる。

参考文献

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013– 2018.[Online], http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[2] 甲藤二郎, “ネットワーク制御と管理(Network-CDN&Application)”,2012

[3] John G. Apostolopoulos, Wai tian Tan, and Susie J.Wee. Video streaming: Concepts, algorithms, and systems.http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf, Sep 2002.

[4] 加藤寧, 西山大樹, 電子情報学会『知識の森』3 群-4 編-5 章

[5] HTTP Live Streaming の概要, Apple Inc., 2014

[6] The MPEG-DASH Standard for Multimedia Streaming Over the Internet, Mitsubishi Electric Research Labs.

[7] DASHEncoder at ITEC/Alpen-Adria-Universität Klagenfurt, [Online], http://www-itec.uni-klu.ac.at/dash/?page_id=282

[8] Benjamin Rainer, Stefan Lederer, Christopher Müller and Christian Timmerer, A Seamless Web Integration of Adaptive HTTP Streaming, In Proceedings of the 20th European Signal Processing Conference 2012, Bucharest, Romania, August 27-31, 2012.

[9] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, “Networking named content,” in Proc. of ACM CoNEXT’ 09

[10] Lixia Zhang, Jeffrey Burke, Van Jacobson, Beichuan Zhang, K.C. Claffy, Christos Papadopoulos, Lan Wang, and Patrick Crowley, “Named Data Networking,” ACM SIGCOMM Comput. Communication, pp. 66-73, Jul. 2014.

[11] CCNx. Ccnx software. [Online], http://www.ccnx.org.

[12] NDNx, [Online], https://github.com/named-data/ndnx

[13] NDN Packet Format Specification 0.2-alpha-3 documentation, [Online], http://named-data.net/doc/ndn-tlv/types.html#type-value-reservations

[14] Node.js, [Online]. Available: http://nodejs.org/

[15] https://www.wattsupmeters.com/secure/products.php?pn=0

[16] “Network characteristics of video streaming traffic,” Proc. ACM CoNEXT 2011, Dec. 2011.

[17] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani, “Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications,”ACM SIGCOMM Conf. on Internet Meas. Conf., pp. 280-293, Nov. 2009.

[18] Test media. [online]. https://media.xiph.org/

[19] Monsoon Solutions. [online]:
http://www.msoon.com/LabEquipment/PowerMonitor/
[20] Takeshi Muto, Kenji Kanai, and Jiro Katto, "Implementation Evaluation of Proactive Content Caching using DASH-NDN-JS", IEEE WCNC 2015, Mar. 2015.
[21] 武藤健史, “NDN 方式を活用した高品質なコンテンツ配信に向けた諸検討”, 修士論文.

謝辞

本研究を行うにあたり、日頃から研究に関して様々なご指導をいただきました甲藤二郎教授に心からの感謝の意を表します。

また、研究等で、アドバイスをしてくださった金井謙治先輩をはじめ、ネットワーク班の先輩方に深く御礼申し上げます。

最後に、研究だけでなく、様々な面でお世話になりました甲藤研究室の皆様と私をここまで育ててくださった家族に、深く感謝致します。

2016 年 2 月 2 日

石津 裕也

発表文献リスト

[1] 石津裕也, 金井謙治, 甲藤二郎, “スマートフォンを用いたビデオストリーミングにおける消費電力量特性評価”, 信学総大, Mar. 2014.

[2] Yuya Ishizu, Jiro Katto, “Evaluation of Energy Consumption in Video Streaming”, 漢陽大学合同ワークショップ, December 2014.

[3] 石津裕也, 金井謙治, 甲藤二郎, “NDNx の輻輳制御方式の改善”, 信学総大, Mar. 2015.

[4] 石津裕也, 金井謙治, 甲藤二郎, 中里秀則, 広瀬真里枝, “Energy-Efficient Video Streaming over Named Data Networking using Interest Aggregation and Playout Buffer Control”, 情報処理学会 オーディオビジュアル複合情報処理研究会, December 2015.

[5] Yuya Ishizu, Kenji Kanai, Jiro Katto, Hidenori Nakazato, Marie Hirose, “Energy-Efficient Video Streaming over Named Data Networking using Interest Aggregation and Playout Buffer Control”, IEEE GreenCom 2015, December 2015.

[6] 石津裕也, 金井謙治, 甲藤二郎, 中里秀則, 広瀬真里枝, “NDN 上のビデオ配信における再生バッファ制御に基づく電力モデル提案と省電力化への検討”, 電子情報通信学会ネットワークシステム/情報ネットワーク研究会, Mar 2016.

[7] 竹中幸子, 石津裕也, 金井謙治, 甲藤二郎, 村瀬勉: “高スループット転送及びバッファリング戦略を活用した移動端末における映像配信省電力化に向けた検討,” 信学技報 CQ 研究会、Mar.2016.

[8] 金井謙治, 小西秀典, 石津裕也, 甲藤二郎: “長期的スループット予測技術に基づく高信頼ビデオストリーミングのためのバッファリング戦略の提案,” 電子情報通信学会 MoNA 研究会, Sep.2014.

[9] 竹中幸子, 石津裕也, 金井謙治, 甲藤二郎, 村瀬勉: “高スループット転送及びバッファリング戦略を活用した移動端末における映像配信省電力化に向けた検討,” 信学技報 CQ 研究会、Mar.2016.

[10] Kenji Kanai, Hidenori Konishi, Yuya Ishizu and Jiro Katto: “A Highly-Reliable Buffer Strategy Based on Long-Term Throughput Prediction for Mobile Video Streaming,” IEEE CCNC 2015, Jan.2015.