動画通信技術の提案と課題

保田 篤輝[*1]
指導教員：吉田 賢史[*2]
Email: yasuda.a.06@akane.waseda.jp

*1：早稲田大学高等学院中学部 2 年
*2：早稲田大学高等学院中学部

◎Key Words　　ストリーミング，伝送技術，圧縮技術

1.　動画通信技術とは

動画データ（映像・音声）をインターネットなどの通信ネットワークを経由して，離れた場所にいる視聴者やサーバーへ届けるための技術の総称である。今回は，その中でもリアルタイム配信（ストリーミング）に着目して説明する。ストリーミングとは，リアルタイム配信のように，データを送りながら同時に再生する技術のことを指す。

現在のストリーミング技術には，主に以下のものが利用されている。

- HLS / MPEG-DASH：広く普及した標準的なストリーミング方式
- LL-HLS：低遅延を実現した HLS の改良版
- WebRTC：リアルタイム双方向通信に特化したプロトコル（ブラウザやアプリで広く採用）

また，データ配信の基盤としては，CDN（Content Delivery Network）がよく利用されている。これは，世界中に配置されたエッジサーバーを経由することで，大容量コンテンツを高速・安定して配信する基盤技術のことであり，「Cloudflare」や「AWS CloudFront」などが代表例として知られている。

また，Netflix の「Open Connect」のように，大規模サービスが独自 CDN を構築している例もある。

2.　ストリーミングに求められる要件

ストリーミング技術には，「安定性」「リアルタイム性」「品質」など，複数の要件が存在する。

今回は，その中でも安定性を重視し，課題と理想について説明する。

今回の場合，「安定性」とは，映像が途切れずにドット抜けや解像度の低下が発生せずに映り続けることを指す。

なお，安定性を優先する分，処理が増えリアルタイム性は犠牲になりやすいと考えられるため，ここではある程度の遅延を許容する前提で話を進める。

用途によって，優先すべき要件は異なる。ストリーミングにおいて，安定性と品質が求められる場面を検討する。

例えば，研究等の動物観察配信や，スポーツ配信，非参加型の動画配信などは，「リアルタイム性」がなくても，「安定性」や「品質」の高さが求められると考えられる。

今回は，研究等の動物観察配信を重視して考えることとする。

現在，安定性を重視したプロトコルとして SRT（Secure Reliable Transport）が知られている。今回の設計においても参考となる技術である。

3.　TCP による安定性の実験と結果

双方向のリアルタイム通信では UDP が主流だが，一方向の動画配信（HLS 等）では TCP による安定通信が既に主流である。

TCP には，不足したパケット（パケットロス分）を再要求するしくみが備わっており，これによってパケットロスをゼロに抑えることができる。

ただし，TCP には深刻な２つの課題がある。1 つめは，回線弱者の切り捨てであり，２つめは，遅延超過時の映像停止である。 回線弱者の切り捨てとは，パケットロスが発生した際に再送要求を繰り返すため，回線が不安定なユーザーを完全に切り捨てる形になってしまう現象である。遅延超過時の映像停止は，一定の遅延域を超えると，画面が黒くなり，その後に映像が一気に早送り再生されるという問題が発生する現象である。

精細な画像を目的とする場合，遅延そのものは許容できるが，遅延が一定の閾値を超えることは避けなければならない。

今回のような動物観察等では，送信側の回線が不安定である可能性も大いにある。そのため，この問題は大きな問題と考えられる。

このような状況に備えて，UDP へ動的に切り替える機構や，パケット数自体を削減して回線弱者を救う手段を検討する必要がある。 しかしながら精細な画像を送る目的を考慮するとパケットロスを調査する必要がある。

画像のように，WSL2 と iperf3 を利用して，通信量を約 1Mbps に制限し，10 秒間 10Mbps で UDP および TCP でデータを送信し，パケットのロス率を計測した。

これを 10 回行ったところ，UDP では，送信された全パケットのうち，約 90%が破棄されるという結果が出た(表 1)。一方，TCP での実験結果では，パケットロス率が 0%であった。また，帯域制限を検知し，自発的に送信速度を約 1Mbps まで強制的に低下させ，データの再送処理も行われていることが確認できた。

4.　パケットの軽量化

4.1　可逆圧縮による対応

回線弱者を救う方法として，まず考えられるのがパケ

表 1 パケットロスの実験結果

プロトコル	パケットロス率	送信速度の挙動
UDP	約 90%破棄†	制限を無視
TCP	0%‡	帯域制限に低下

†図 1 参照。　‡図 2 参照

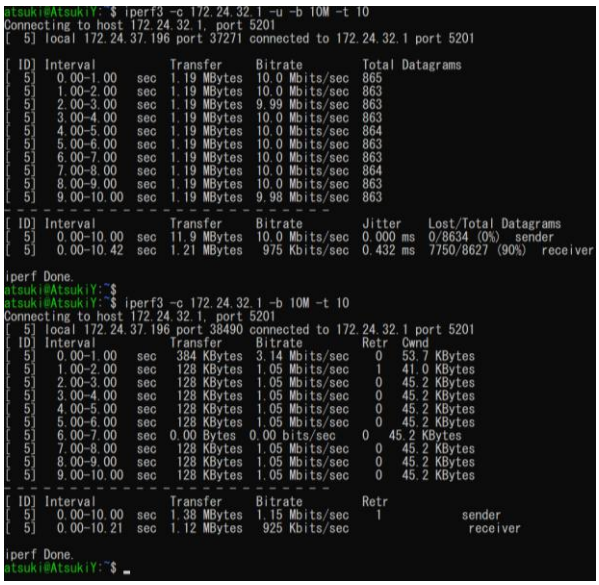

図 1 送信側の UDP/TCP での計測画面

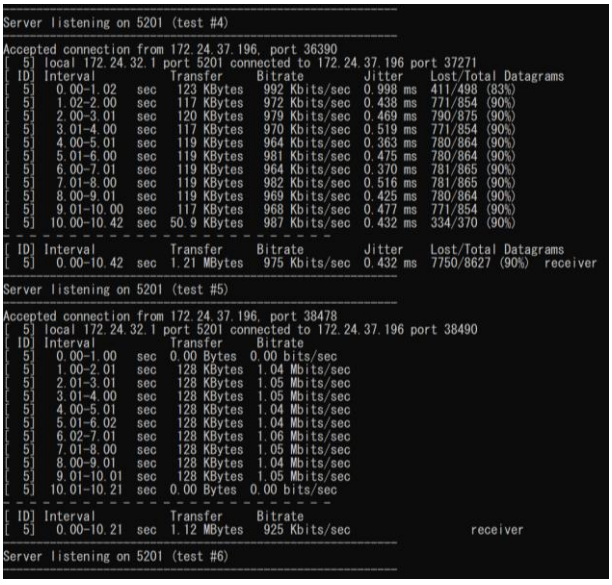

図 2 受信側の UDP/TCP での計測画面

ットの軽量化（圧縮）である。今回は「安定性」と「品質」を重視しているため，データを完全に復元できる可逆圧縮を採用する必要がある。

可逆圧縮は現在でも文章データ等で広く利用されているが，映像データへの適用においては圧縮率が低いという課題がある。また，映像通信において，圧縮を復号する処理によって，時間もかかるので，そこも課題である。

4.2 送信部分の軽量化

可逆圧縮による軽量化にも，限界が存在している。

そこで，今回のような研究等の動物観察配信は固定カメラのように，ある一定のところから撮影し，画面の変化が少ないことに注目した。変化のあった部分だけを送信し，受信側では前のフレームに上書きして表示する。これを繰り返すことで，変化分だけを送信することができ，また，軽量化ができる。

この方法は，後に知ったが，MPEG や H.264，H.265 といった現在の標準的な動画圧縮技術（フレーム間予測圧縮）の根本原理であった。

ただし，必要分だけ送信であっても，最初の一回や，リロード直後はすべてダウンロードする必要がある。そこで，もとから仮の色を入れておき，それで画面を埋め尽くしたうえで，同じような計算を実行するのがいいのではと考えた。撮影された 1 度目のデータを RAM 上で確認し，その中から一番多いもので埋め尽くす処理を，繰り返しでいいので，非常に少ないパケットで送信することができる。

しかし，このような技術を実行するには処理速度が課題となる。つまり，画像センサーから送られたデータを RAM に保存し，前回送られた RAM 上のデータと比較し，差分のデータを送信する。待機列の次のフレームデータを送信後の RAM 領域に上書きする。待機列にフレーム情報を送信し，RAM にロードした待機データを待機列から削除する。この工程は，処理に時間がかかり，品質も担保するため，30fps で再生することを考えると，この処理方法は難しいと考えられる。

4.3 処理方法

そこで，YouTube などの別の動画配信サイトではどのような処理をおこなっているのか調査した。バイナリ値を送信し，それをクライアント側で復号し，その情報をもとにクライアントで映像を作成しているのか，それとも別の方法なのか検討する。しかしながら，結論としてはおおむね実験前の予測通りであった。

映像を受け取り，デコードを行い，カラースペース変換を行った後，GPU チップの中にある専用のデコード回路，ハードウェアデコーダーを利用して映像化処理をおこなっていた。このハードウェアデコーダーのおかげで，スマホでも快適に映像の処理を行えていた。GPU を使用せず CPU だけで処理するのは，高速な CPU を用いても処理に負荷がかかり，高精細な処理の実現は難しいと推測する。

それでは，そのデコーダーロジックはアプリを使用せず web で視聴する場合，どのように端末に処理方法を伝えているのだろうか。

調査したところ，Safari や Chrome，Firefox などの web ブラウザに，もとからデコーダーロジックを組み込んでいる。そのため，ブラウザをインストールした時点で，ユーザーが気づかぬうちにデコーダーロジックを端末内にダウンロードされている仕組みである。

過去のブラウザはこのような機能を有しないため，専用のプラグインをダウンロードする必要もあった。しかし，これでは主流なデコーダーロジックしか利用できない。

独自に開発したデコーダーロジックをブラウザで使用したい場合，専用のプラグインを導入する手間が発生する。今回のような研究的な実験を考えると，問題は生じないが，できるだけ手間を増やしたくないので，ロジックをダウンロードさせる方法を考えたい。

そこで考えたのは Cookie の活用である。Cookie にデコーダーロジックをダウンロードさせる機能をもたせられないか検討した。

しかし，Cookie には最大約 4KB までしか保存できない。この容量はデコーダーロジックをダウンロードさせるのには少なすぎる。ダウンロードを必要としないデコーダ

ーロジックを調べたところ Web Assembly (WASM) という技術で実現可能であると考えた。

Web Assembly は，ブラウザ上で C/C++/Rust などのネイティブコードを JavaScript から呼び出せる技術で，これを利用することで，デコーダーロジックの導入を実現可能である。

また，システムのサイトに再訪問したときは，Service Worker と Cache API で，デコーダーロジックの再ダウンロード不要となる仕様が提供されている。Cache API という仕組みもあり，デコーダーロジックのダウンロードを行う際，Service Worker という機能が働き，サーバーへのリクエストを割り込み命令として受け取ることが可能となる。この機能によりキャッシュ済みのデータとして Web Assembly がユーザーの操作不要で即送信可能となる。

Cache API は，Cookie とは違い，デバイスのストレージ容量に依存するため，Cookie のように，最大容量に悩まされることもない。

しかし，大きな弱点も存在しており，Web Assembly は，GPU ハードウェアデコーダーが利用できない。GPU ハードウェアデコーダーが利用できないことで，CPU ソフトウェアデコーダーに頼るしかなくなり，ネイティブのブラウザデコーダーよりも重くなってしまう。

また，高解像度・高フレームレートの動画には性能が足りない場合もある。研究等で利用するなら，スペックも十分に担保可能だが，実用レベルで配信する場合，一般人は見られないという事態が発生してしまう。そのため，別の方法も考える必要がある。

4.4 AI の活用

送信するフレーム数，画素数を減らせば，その分だけパケットを小さくすることができる。ただし，品質を求めているので，それは難しいだろうという結論に至った。

そこで，AI の活用を検討する。大手 GPU メーカーである Nvidia の RTX シリーズでは，DLSS という，AI を利用して，間のフレームを生成し，それを再生することで，フレーム数を大幅に増やすことができる技術を搭載している。最近では DLSS 4.5 が発表され，DLSS 4.5 以前に比べて性能が大幅に向上し，違和感のないフレーム生成として少し話題になった。

しかしながら，今回のような観察実験に AI を組み込むことは，実際の様子ではないものを見ることになるので，利用は慎重に検討する必要があると考えられる。

5. 接続方式の検討

5.1 P2P 接続の特徴

1 対 1 通信を主に行う一部のサービスでは，P2P (Peer-to-Peer) 接続が採用されている。

これは 2 つのクライアントが直接通信する方式であり，サーバー側の負荷が小さく，仲介が不要な分，低遅延を実現できる。また，仲介サーバーが存在しないため，サーバー障害の影響を受けずに通信を維持できるという点で，安定性の観点からも優れている。

5.2 1 対多数への対応課題

しかし，今回の主な対象である動物観察配信などの用途では，1 対多数の構成が想定される。

そのため，P2P 接続をそのまま適用することは難しい。サーバー側の負荷を抑えながら，1 対多数に対応し，かつ安定性に特化した配信サービスの設計が必要となる。

6. 流暢な再生のためのバッファリング

6.1 TCP ではなく UDP が使われてきた理由

これまでの映像・音楽再生に TCP ではなく UDP が採用されてきたのは，主にリアルタイム性の欠如と流暢な再生ができないという 2 つの理由による。

リアルタイム性の欠如は，TCP はコネクション確立とパケットロスの再送処理が必要なため，リアルタイム性が損なわれる。流暢な再生ができないことについては，第 3 節の実験結果が示す通り，TCP は失われたパケットがすべて揃ってからしか映像を表示しないため，回線が細い環境やパケットロスが発生する環境では映像が完全に止まってしまう。一方，UDP は再送処理を行わずパケットを破棄するため，パケットロスがあっても一瞬の乱れで済む。

6.2 バッファによる解決

そこで，一時的にデータを蓄積する受信バッファ（RAM バッファ）の導入を提案する。

前述の「10 秒〜1 分程度の遅延許容」はここに起因している。例えば，10 秒分のデータを保持できるバッファを設けた場合，最初の 10 秒間は映像が再生されないが，その後は常に 10 秒分のデータが蓄積された状態で再生が進む。送信側で一時的な遅延やパケットロスが発生しても，バッファ内のデータを消費している間に再送処理が完了するため，映像が途切れることなく流暢な再生を維持できる。

6.3 バッファによる再生の挙動

実際に，動画再生プラットフォームとしてよく知られる，YouTube とブラウザに内蔵されているデベロッパーツールを利用して，バッファによる再生が本当にうまくいくのかを次の手順で確認した。

第 1 にインターネット環境で動画を数秒再生する。第 2 に，デベロッパーツールを用いて，通信を完全に遮断する。第 3 に JavaScript を用いて，動画を再生してから停止するまでの時間を計測した（図 3）。

これを 5 回行った結果，バッファによる再生で維持できた時間の平均は，約 24.25 秒であった(表 2)。

このように，バッファを設けることで，インターネット

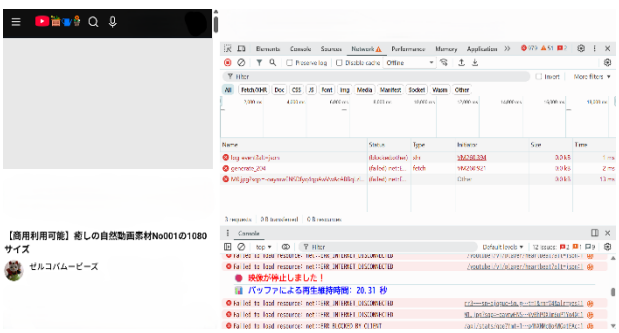

図 3 計測の方法とその様子

表 2 バッファ計測の結果

	再生維持	画質	再生速度
結果	約24.25秒	1080p	標準

が不安定な状況でも，状況を立て直す程の猶予時間を確保することが可能であるとわかった。

また，その間も流暢な再生を実現することができていた。

7. 結論

結論としては，今回のような貴重な研究対象等に対する安定性を重視したストリーミング技術に関して，TCP を利用した安定的な通信方法の利用と，バッファを利用することによる安定的な通信技術と，効果的な可逆圧縮によるパケットの軽量化を用いることで，問題を解決できると分かった。

また，これらはすでに FFmpeg 等で実現がされており，かなり安定したストリーミングがすでに確立されつつあることも分かった。

しかし，可逆圧縮などの，送信側のスペックが必要になってしまうなどの問題点もあると考えた。

参考文献

(1) マスタリング TCP/IP 入門編 第6版(出版ページ):
https://www.amazon.co.jp/マスタリング TCP-IP―入門編―第6版-井上-直也/dp/4274224473

(2) Rakuten Mobile「ストリーミングとは？ダウンロードとの違いや仕組みを解説」:
https://network.mobile.rakuten.co.jp/sumakatsu/contents/articles/2022/00041/ (参照 2026-6-8)

(3) AWS「CDN(コンテンツ配信ネットワーク)とは何ですか？」:
https://aws.amazon.com/jp/what-is/cdn/ (参照 2026-6-8)

(4) SimulTrans「SRT ファイルとは」:
https://www.simultrans.com/ja/blog/what-is-an-srt-file (参照 2026-6-8)

(5) iperf3: https://iperf.fr/iperf-download.php (参照 2026-6-8)

(6) WSL2:
https://learn.microsoft.com/ja-jp/windows/wsl/install (参照 2026-6-8)

(7) IT 用語辞典「ハードウェアデコード【hardware decoding】」:
https://e-words.jp/w/ハードウェアデコード.html (参照 2026-6-8)

(8) Wikipedia「WebAssembly」:
https://ja.wikipedia.org/wiki/WebAssembly (参照 2026-6-8)

(9) Nvidia: https://www.nvidia.com (参照 2026-6-8)

(10) Coincheck「P2P(ピアツーピア)とは？仕組みやメリット，活用事例を紹介」:
https://coincheck.com/ja/article/577 (参照 2026-6-8)

(11) YouTube - ゼルコバムービーズ「【商用利用可能】癒しの自然動画素材No001 の 1080 サイズ」:
https://www.youtube.com/watch?v=W_OK4TLbvYs (参照 2026-6-6)

(12) FFmpeg: https://ffmpeg.org/ (参照 2026-6-8)