
ライブ配信のなかみ
Inside Live Haishin

ライブ配信のなかみ

Inside Live Haishin

あれくま

目次

第1章 はじめに

今日このごろは回線速度も上がり、機器の処理速度向上もあって個人がインターネットで動画のライブ配信をすることも一般的になってきました。

配信サービスも新しく出来ては消えゆくこのご時勢。さぞかし新しい技術開発も進んでいるのかと思いきや、使われている技術の移り変わりは意外にもゆっくりとしたものです。

この本ではそんな昨今の動画配信、その中でもライブ配信を主眼に、使われる技術について解説します。

第2章 ライブストリーミングとは

この章では、まずライブストリーミング配信についてざっくりとした話を解説します。

1. ストリーミング再生

ネットワーク上にある動画を視聴したいとします。したいですよね？うん、みんなしたいはずだ。

ネットワーク上にあるのでまずはダウンロードしないと見れません。動画をダウンロードして視聴するには大きく分けると二つの方法があります。

1. 全部ダウンロードしてから再生を始める
2. 頭からダウンロードしつつ落とせたところまで再生してしまう

全部ダウンロードしてから見るのはわかりやすいですね。ダウンロードが終わってしまえば普通にローカルにある動画を視聴するのと同じです。早送りや巻き戻し、というか好きなところから見ることも可能です。

簡単で分かりやすいこの方法の問題点の一つとして、全部ダウンロードが終わらないと視聴が始められないことがあります。数秒や数分程度の小さな動画ならいまどきの回線速度なら数分もかからずダウンロードが終わるかもしれませんが、2 時間の映画なんぞをダウンロードしようと思ったら数分でとはいきません。少なくとも数十分は待つ覚悟が必要でしょう。もう一つ問題点を挙げるならば、全部ダウンロードできるだけのストレージ容量も必要となります。多少長くても動画一本の容量はたかが知れてますので、PC のディスクがいっぱいになることはあまり無いと思いますが、スマートフォンやタブレットのスマートフォンのストレージだとちょっと躊躇われるかもしれません。

頭からダウンロードしつつ落とせたところまで再生開始する方法では、全部落とすまでの待ち時間を回避できます。最低限再生できそうな分だけダウンロードできたら再生し始めてしまいますので、上手くいけば 1 分も待つことなく視聴が始められるでしょう。普通にやるとストレージ容量は最終的には全部ダウンロードするのと同じだけ必要になってしまいますが、再生し終わった分は削除するように一工夫すると必要なくなります。削除してしまうと巻き戻しはできなくなるのでちょっとデメリットもありますが。

こちらの問題点としてはダウンロード速度が再生速度より速くないと、途中で止まってしまう点があります。始まるまでの時間は短いけど途切れ途切れではつらいですね。また、まだダウンロードしてないところは再生できないので早送りはできません。そのままなら

巻き戻しはできるのですが、ストレージ節約のために再生し終わったところは消すようにすると巻き戻しもできなくなります。

このように一長一短あるのでどちらも使われるものではありますが、特に後者のダウンロードしつつ再生する方をストリーミング再生と呼びます。全部ダウンロードする方はなんて言うのかというと一般的には特別な名前は付いてなさそうです。普通すぎるので特に何とは言わないんでしょうか。とにかくダウンロードしてるそばから再生していくのがストリーミング再生です。

2. ストリーミング配信

ファイルや動画をダウンロードできるようにするのを配信と呼びますが、ストリーミング再生用に配信するのを特にストリーミング配信と呼びます。

普通にダウンロード用に配信されているものをストリーミング再生するのもできなくはないのですが、ストリーミング再生を意図して配信する場合にはストリーミングの弱点を補えるような機能を提供したりします。

ストリーミング配信で提供されがちな機能には次のようなものがあります。

- ビットレート (bitrate) の切り替え
- シーク (seek) の実現

ビットレートというのは単位時間あたりのデータ量のことです。ビットレート高い＝動画 1 秒あたりのデータ量多い＝綺麗、ということでだいたい合ってます。まあ必ずしもビットレートが高ければ綺麗とは限らないんですが、1 秒あたりのデータ量が多ければその分綺麗にする余地があるということなのでだいたい高ければ綺麗です。

先程も書いた通り、ストリーミング再生ではダウンロード速度が再生速度を上回らないと視聴が止まってしまってイライラするはめになります。1 秒あたりのデータ量が多ければ 1 秒あたりにダウンロードできる量 (＝回線速度) も多くないといけません。つまり回線速度がビットレートより高くないと再生が止まってしまいます。

回線速度が低くてダウンロードが間に合わず止まってしまう場合はどうしましょう。もちろん回線を速くすればいいんですよね！……ってそう簡単に速くできるなら苦労しないのですが、回線速度は往々にしてお金に直結するので簡単には増えません。仕方ないので普通はビットレートの方を下げます。画質や音質は犠牲になりますが、詰まらずに視聴できる方が快適です。

完全ダウンロードの配信でも複数のビットレート別ファイルを用意することはありますが、ストリーミング配信の場合は途中でのビットレート切り替えにも対応することが多いでしょう。その時々の回線速度に合わせてビットレートを切り替えることによって、詰ま

らない範囲でなるべく高画質の動画を視聴できるようにします。

もう一つはシークの実現です。シークとは直訳すると捜すとか捜し求めるなんですが、これは単に早送り巻き戻しの対応のことです。まだダウンロードできてないところへの早送りは普通に考えるとできないですが、まだダウンロードできてないところをすっとばして見たいところからダウンロードしてしまえば実現できます。逆に再生し終わったので捨ててしまった部分への巻き戻しも、そこからまたダウンロードを始めることで実現できます。

一言で言うと簡単そうですが、実際やろうとするとこれがなかなか難しいものです。というのも、シークしたい場所というのは時間で指定します。CM とばしたいから 90 秒スキップ、とかやりますよね。しかしダウンロードはバイト単位で行っています。さて 90 秒後とはいったい何バイト先なのでしょうか……?

シークの実現方法はいろいろありますが、一つにビットレートを一定にしておくなどがあります。たとえばビットレートが 800kbps の動画であれば 1 秒間がだいたい 100 キロバイトになるはずなので、90 秒後は 9000 キロバイト先を見ればいいだけです。まあきっかり 9000 キロバイト先になるとは限りませんが、だいたい合ってればそんなに問題は出ないでしょう。

他に時間単位の位置とバイト単位の位置を関連付ける表を作っておくこともできます。たとえば 1 秒毎にバイト位置を記録しておけば、90 秒後が何バイト目かはすぐ出せます。表の分だけ余計なデータが必要になりますが、動画本体に比べれば些細なものです。ビットレートが一定にならない場合にはこちらの方を使います。

3. ライブだからなんだって言うの

長々とネットワーク上の動画をダウンロードして視聴する方法について書きましたが、そういえばこの本はライブ配信 (生配信) について書くんだった。

ライブ配信の場合には全部ダウンロードしてから視聴というのはできません。今現在作られていくデータを前もって全部ダウンロードするのはできませんからね。録画しておいてあとからゆっくり見るのはもちろんできますが、それはライブ視聴とは言わないでしょう。俺はリアルタイムで見たいんだ。

そんなわけで、ライブ配信の場合は必然的にストリーミング配信になります。ストリーミング配信自体については先に書きましたが、そのなかでもライブ配信ではまた少しずつ違ってくる点があります。ここでは大きく二点挙げましょう。

- 事前処理ができない
- 実時間以上かかる処理ができない
- 遅延を減らしたい

三点あるな？それぞれ関連する話でもあるので細かいことは気にしないようにしましょう。

ライブ配信の場合は、単にストリーミング配信するのと違って事前処理ができないという制限が付きます。事前処理というのは配信前に全部データが揃った状態でやる処理ですね。事前処理では動画を最初から最後まで見てから圧縮することで綺麗に圧縮する 2 パスエンコードや、全体の音量レベルを調整しておくとかいうのができます。ライブ配信の場合は最後まで見ての処理というのが当然できないので、事前処理はできません。

また、ライブ配信では実時間以上かかる処理もできません。1 秒分の動画を圧縮するのに 1 秒以上はかけられないということです。これを突破してしまうと処理待ちが発生し、視聴が止まってしまいます。止まるのを回避するためにコマ落ちさせて処理することもできますが、当然カクカクになるので残念な気持ちにはなります。だいたい処理量を増やせば増やすほど綺麗に圧縮できるので、ライブ配信では高速な処理ができるプロセッサを使うか、画質をある程度諦めるかする必要があるでしょう。これがライブ配信でなければ、事前処理しておいて処理済みの動画を配信すればいいので、いくらでもゆっくり時間をかけて処理することができます。実際のところ事前処理するにしてもいくらでも時間をかけるわけにはいかないので速いに越したことはないのですが、それでもライブ配信ほどリアルタイム性が要求されないので楽に画質を上げることができます。

遅延を減らしたいというのは制限というより要求なのですが、ライブ配信ではできれば遅延を減らしたくなります。遅延というのは画面が出来てから動画データにして視聴者に届いて再生されて見られるまでの時間です。遅延が短ければ短い程、配信側の出来事が視聴者にすぐ伝わります。せっかくのライブなのですからリアルタイム性は欲しいですし、いまどきは視聴者との双方向性も重要でしょう。コメントや投票などはすぐ反映されてほしいですよね？数分もあるとだいぶ遅いと感じますし、できれば数秒の遅延、いやなんなら 1 秒未満の遅延で済んでくれれば最高です。しかしながら遅延はなかなか減らせません。

遅延の原因の一つは通信にかかる時間があります。インターネットを介しての通信は近いところでも数十ミリ秒、ちょっと遠かったり通信状況が悪いと数百ミリ秒になります。視聴するだけなら片道ですが、コメントを送ってからその反応が視聴できるまでとなると往復で倍かかります。といっても余程遠くでなければ何秒もはかかることは無いでしょう。しかしながら、どうしても短くするのは難しいところではあります。

より大きな要因としては各種バッファの分があります。動画の処理はある程度のデータを一旦溜めてから行うため､溜めてる時間分の遅延はまぬがれません。通信遅延と違ってバッファを少なくすることである程度短くできるところですが、短くすると処理時間のぶれを少なくする必要が出てきます。動画データは画面の内容などによってデータ量が変わるため、処理時間にぶれが出てくるのですが、バッファが短いと許容されるぶれも短くなってしまいます。

たとえば 10 秒のバッファを持って処理する場合、10 秒分の動画データを平均 8 秒± 1

秒で処理できれば、10 秒くらいの遅延はありますが間に合います。よっしゃじゃあ 1 秒分の動画データだったら平均 0.8 秒± 0.1 秒で処理できて遅延 1 秒じゃねーか！と言いたくなりますが、処理時間はデータ量にそのまま比例しない部分があるため、1 秒分の動画データが平均 0.8 秒± 0.5 秒くらいでしか処理できなかったりします。最悪 0.8 秒 +0.5 秒で 1.3 秒になると、おっと 1 秒を突破してしまいました。間に合わないため動画再生が止まったりカクついてしまいます。単純に短くしていいものでもないですね。

このバッファは動画の処理だけでなく、送受信のバッファとしても存在します。インターネット回線は混み具合などにより通信速度にものすごいぶれが発生します。さっき 100 ミリ秒で届いてたデータが、次の瞬間にはいきなり届くのに 1 秒かかるようになり、また 100 ミリ秒で届くように戻ったりします。これもある程度溜め込んでから再生することでぶれを吸収することができます。そのぶん遅延が発生してしまいますが、通信バッファを短くしすぎると通信がちょっと詰まるだけで動画が止まりがちになりイライラするはめになるでしょう。

このようにただ配信するだけならなんとかなるストリーミング配信も、ライブ配信となるだけで時間との戦いが始まってしまうのです。

第3章 動画データの基礎知識

ここからはもうちょっと中身の話をしていきましょう。

みんな知ってるようでよくわかってなかったりする動画データのお話です。

1. 動画データの構造

動画データはどうやって出来ているでしょうか。

まずは映像は必要です。そりゃ動画データと言ってるのでそうでしょう。映像は一定時間単位で画像を並べたものになっています。

最近はあまり見なくなったFlashアニメや、一行に流行る気配のないSVGアニメーションや、ゲームではおなじみの3Dモデルのアニメーションなんかは時間に沿ってパラメータで画像を変形させていくものですが、一般的な映像データではパラパラマンガの要領で一定時間毎に画像を切り替えていくものがほとんどです。以降解説するデータもパラパラマンガ形式だと思ってください。

多くの場合音声も欲しいです。昔のサイレント映画やGIFアニメなんかは音声が付いてませんが、まあ普通動画データと言えば音声付きを期待するでしょう。音声は空気の振動ですので波形データとして表せます。音声データはある瞬間の振幅を記録して、それを時間毎に並べていくことで波形を作ります。この方法は一定時間毎に振幅を切り替えていくパラパラマンガ形式と思えばわかりやすいでしょう。そんなことないですね。わかりづらいですね。でもまあ対比としてはそんな感じです。この音階の音をこのタイミングでこの時間だけ流す、といった音声のデータもありますが(MIDI等)、一般的な動画ではほぼ使わ

れないのでここでは解説しません。

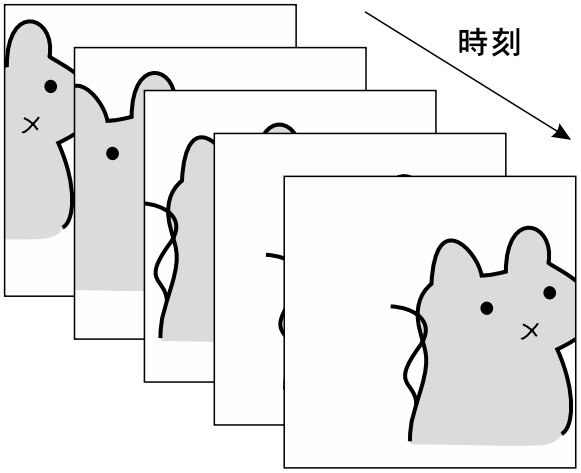


図 1. 映像データのイメージ

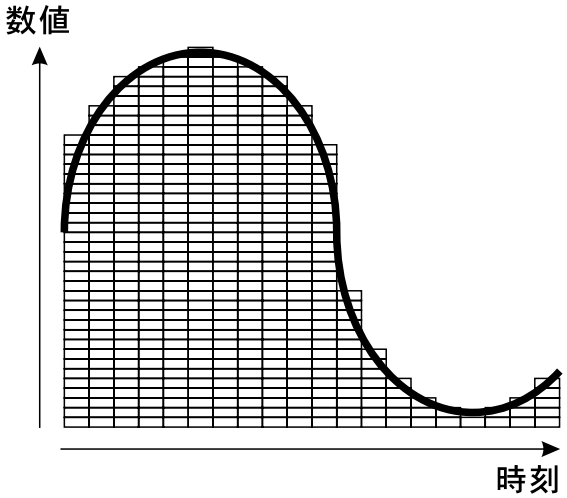


図 2. 音声データのイメージ

あとライブ配信ではあまり使われないと思いますが、字幕データが載ったりします。字幕だけでなくなんらかの付加データ (データ放送的なもの) を載せたりもできます。どうやって載せるのかやどうやって再生するのかは特にライブ配信だと難しかったりしますが、まあできることはできます。

他に直接動画中で視聴するものではない、タイトルや作者、演者、概要などといった付加情報が入ることもあります。こういった付加情報はメタデータと呼ばれます。

これらをひとまとめにしたものがみなさんのお手元に届くことになります。

動画データには映像と音声などがひとまとめに入りますが、このひとまとめにすることを多重化 (Multiplexing)、略して Mux(マックス) と呼びます。逆にひとまとめにされた動画データから映像や音声を取り出すのを逆多重化や多重分離 (どちらも Demultiplexing)、

略して Demux(デマックス) と呼びます。多重化するプログラムは Muxer(マクサー)、多重分離するプログラムは Demuxer(デマクサー) ですが、分離するので Splitter(スプリッター) と呼ぶこともあります。べつに知らなくても困らないんですが、動画ファイルの中身をいじろうとした時には Mux や Demux,Splitter なんかは度々出てくる言葉なので覚えておくと役立つでしょう。

ちなみに多重化という名前からは、映像と音声をひとまとめにするというより、複数の音声をひとまとめにしたり複数の映像をひとまとめにするというのを想像できますが、そういうこともできます。多くの動画ファイルには複数の映像を同時に入れたりもできます。何に使うのかというと、多視点のビデオ切り替えや副音声の実現に使えます。こうなると多重化って感じがしますね。

2. データの圧縮

映像データや音声データはそれぞれある瞬間のデータをパラパラマンガ形式で並べていくのですが、何も考えずにそのまま並べると膨大な量になります。

音声データは CD くらいの音質ですと 16bit のデータを 44.1kHz(1 秒間に 44100 個) で並べます。1 秒間に 88,200 バイト、ビット単位で言うと 705,600bps です。700kbps くらいならいまどきの通信回線ならそこまで大きくは……おっと、これはモノラルデータでした。ステレオデータになると 2 倍なので 1,411kbps、ざっと 1.4Mbps です。無理な量ではないですが、小さくもないですね。モバイル回線では混んでると厳しいかもしれない量です。

映像データは DVD くらいの画質ですと 1 ピクセルあたり 16bit で 720 × 480 の画像を 30fps(秒間 30 コマ) で並べます。1 秒間に 20,736,000 バイトです。実際には DVD だともっと間引いた画質なのでこの 1/4 くらいだったりしますが、それでも 1 秒間に 5,184,000 バイトです。圧倒的にでかいですね。ビット単位で言うと 40Mbps です。いまどきの光回線なら 1Gbps だとかモバイル回線でも 440Mbps とか出るんだぜ！余裕！と言いたいところですが、実際にそんな理論値が出ることはないので、40Mbps では光回線でも回線が混んでなければなんとか……といったところでしょう。モバイル回線だと送受信できたとしてもあっという間に使用量制限を食い潰してしまいます。しかもこれは DVD 画質なので、地デジくらいの画質にしたいなぁと思ったら 4 倍以上になります。もはやリアルタイムでは無理がありますね。

とまあ、これらの映像や音声データはそのまま保存するにしても流すにしてもかなりの量なので、普通は圧縮して扱います。

データの圧縮には大きく可逆圧縮と不可逆圧縮があります。

可逆圧縮は劣化がない圧縮方法で、zip ファイルなんかに使われます。画像だと PNG ファ

イルや、音声だと FLAC といった圧縮方法があります。劣化がないというのは圧縮解除すると完全に元に戻せるという意味です。元データと全く同じ綺麗さを保てるのですが、完全に元に戻せる必要があるのでいくらでも小さくするということはできません。またデータの内容によっては小さくなりづらかったり、圧縮してみないとどれだけ小さくなるかわからないので、狙ったサイズにするようなコントロールはできません。

不可逆圧縮は劣化を許容する圧縮方法です。画像だと JPEG ファイル、音声だと MP3 等が該当します。劣化というのはデータを間引いて小さくするという意味です。間引いてしまう以上、元のデータと同じにはならないのですが、何も考えずに間引くわけではありません。なるべく目立たないような箇所を選んで間引くため、あまり劣化を感じさせずにデータ量を減らすことができます。また、間引き具合を制御することで目標とするデータ量に近づけるように圧縮することができます。あまり小さくしようとすると目に見えて汚なくなったりカクカクしたりしますが、細い回線向けには多少汚くても最低限見れるように少ないデータ量、速い回線向けには綺麗な大きめのデータ量と使いわけもできるのがメリットです。

映像ではほぼ不可逆圧縮が使われます。映像データは可逆圧縮してもまだデータ量が多すぎるので扱いづらいようです。可逆圧縮ができたとしても、配信向けにはデータ量のコントロールができる不可逆圧縮の方が有利ですね。

音声では可逆圧縮も現実的なため両方とも使われますが、動画に入れる場合は音声にも不可逆圧縮が使われることが多いです。可逆圧縮では圧縮されたデータ量を一定にするためにちょっとサイズが大きめになりますので、容量に余裕があるときには使われるようです。

3. 動画ファイルの構成

動画の構成についてなんとなくわかったところで、今度は実際の動画ファイルについて見ていきましょう。とりあえず配信するかどうかに限らず、普通の動画ファイルについて考えます。

さて、動画ファイルにも用途などによっていろいろな形式がありますが、どんな形式があるでしょう？ WMV、MP4 ？ MPEG-4 ？ H.264、AVC、WebM、mp3…は音楽だっけ？ MPEG-2 ってなんだっけ、m2ts とかいうのもあったような……。良く聞くようなのや実際に見るファイルの拡張子なんかを挙げてみましたが、実はこの中にはファイル形式でない物も混ざっています。動画データでよくわからなくなるコンテナとコーデックの罠です！

動画ファイルは、入れものである動画ファイル自身の形式と中に入れる映像や音声の圧縮形式が独立していて、同じ形式 (同じ拡張子) のファイルでも、映像や音声の圧縮形式が違うことがあります。

画像ファイルで言えばJPEGファイルですと、JPEGファイルの中身はJPEG形式で圧縮された画像で、PNGファイルの中身はPNG形式で圧縮された画像と簡単です。一方で動画だと、MP4ファイルの中身にAVC形式で圧縮された動画とMP3形式で圧縮された音声を入れたり、同じMP4ファイルなのにHEVC形式で圧縮された動画とAAC形式で圧縮された音声が入ったりということができます。

動画ファイルはファイル形式だけでなく、中に入っているデータの圧縮形式も気にする必要があるんですね。このおかげでいろんな形式が出てきて話がごっちゃになりがちです。さらにおそろしいことに、全く同じ形式なのにいくつも名前があったり、名前が似てるのに全然関係ない形式だったり、名前が同じなのに別物だったりというのが存在するため大変分かりづらいことになっています。

ここからは各形式の名前と種類を覚えて区別できるようにしていきましょう。これを覚えるとごっちゃになっている人と話をした時に、お、こいつ分かってないな？とちょっと優越感に浸れます。ただし露骨に出すと嫌われるでしょう。

まず、動画ファイル自身の形式のことはコンテナと呼びます。中に入ってる映像や音声の圧縮形式のことはコーデックと呼びます。

動画ファイルの形式はコンテナ、中の圧縮形式はコーデックです。OK？特に大事なので繰り返しましたよ。構成としては図のような形になります。

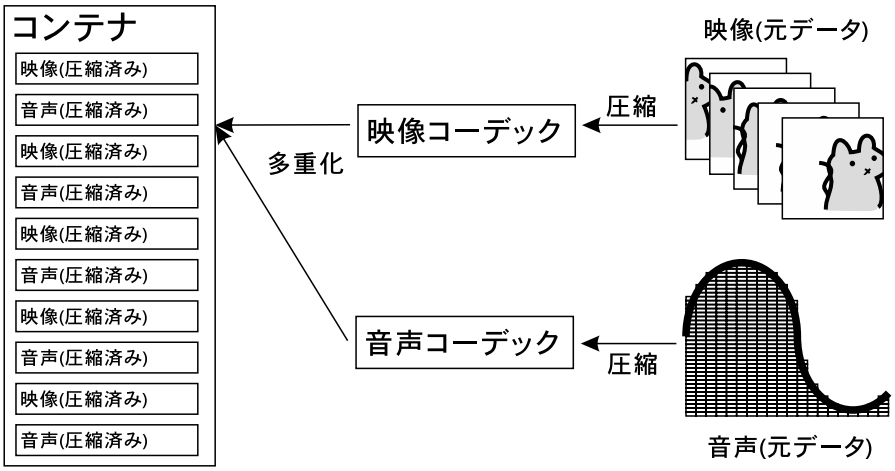


図3. 動画の構成

基本的にはコンテナは拡張子で判別できて、中身の圧縮形式であるコーデックはファイルの中まで見ないと分かりません。

とりあえず拡張子で簡単に判別できるコンテナについて良く見かけるもの挙げていきましょう。

名前	拡張子
AVI	.avi
Advanced Streaming Format (ASF)/ WMV	.asf, .wmv, .wma
MP4	.mp4, .m4v, .m4a, .3gp
MPEG-2 TS	.ts, .m2ts
MPEG-2 PS	.mpg, .mpeg, .m2p, .vob
Matroska/WebM	.mkv, .webm
Flash Video	.flv
Ogg	.ogg, .ogv, .oga
Quicktime	.mov

表 1. コンテナと拡張子

もはやあまり使われないのも挙げましたが、なんとなく見たことがあるのがいくつかはあるんじゃないでしょうか。

コーデックについても挙げておきましょう。これらは普通コンテナの中に入っているので、単体のファイルとしては出てきません。

- WMV 7 〜 9, WMV 9 Advanced, VC-1
- WMA
- MP3
- MPEG-2 Video, H.262
- MPEG-4 Video, H.263
- AVC, H.264
- HEVC, H.265
- AAC
- Vorbis
- Theora
- VP6
- VP8
- VP9
- Opus

こっちもあんまり使われなくなった物も挙げてますが、いくらか聞いたことがあるものもあるでしょう。

この先ではそれぞれのコンテナやコーデックについて解説していきます。

3.1. いろんな MPEG

動画の話では必ずと言っていい程出てくる言葉「MPEG」ですが、これは何を指してるのでしょうか？

まず、単に MPEG と言った場合、Moving Picture Expert Group の略です。直訳すると動画専門家集団でしょうか。強そう。動画に関するいろんな国際規格を作ってる人達です。実際強い。

この人達が作った規格が MPEG-1、MPEG-2、MPEG-4、MPEG-DASH といった規格です。ハイフンの後ろに数字か文字が付きます。これらの規格は MPEG- なんちゃらが正式名称で、Moving Picture Expert Group-1 とかの略ではないようです。

この MPEG-1、MPEG-2、MPEG-4……といった規格なんですが、実はこれらは単一の規格ではなく、いろんなパートに分かれた規格群です。よく聞く物では、MPEG-2 は主にデジテルテレビ向けの動画に関する規格、MPEG-4 は主にインターネット向けの動画に関する規格が規定されています。

中にはコンテナの規格について書いたパートやコーデックの規格について書いたパートが含まれます。ですので、単に MPEG-4 と言っただけでは具体的には何を指しているのかわからず、たいていはパート番号やパート名を付けて表します。

MPEG なんちゃらで混乱しないよう気を付けたいものには以下のようなのがあります。

名前	**意味**
MPEG-2 TS (Transport Stream)	コンテナ
MPEG-2 PS (Program Stream)	コンテナ
MPEG-2 Video	映像コーデック
MPEG-4 Video	映像コーデック
MP4	コンテナ
MPEG-4 AVC	映像コーデック。MPEG-4 Video とは別物
MPEG-4 ALS	音声コーデック。MPEG-4 Audio の一部
MPEG-4 SLS	音声コーデック。MPEG-4 Audio の一部
MPEG-DASH	ストリーミングプロトコル

表 2. いろんな MPEG

似てる名前が沢山ありますし、MPEG-2 や MPEG-4 それぞれの中でも複数のコンテナやコーデックが規定されていますので、単に「MPEG-2」や「MPEG-4」だけでは区別がつけら

れない場合があります。判別できる程度までしっかり言うようにしましょう。

4. コンテナについて

コンテナとは先にも述べた通り映像や音声データなどの入れ物です。動画ファイルのことをコンテナと呼ぶと思ってほぼ間違いないです。

コンテナにもいろいろな形式があります。なんでいろいろあるのかというと、それぞれやりたいことが少しずつ違うからでしょう。コンテナにも向き不向きがあったりします。

4.1. MP4

最近だと一番良く見る動画ファイル形式でしょう。

MP4 は MPEG-4 の Part 14 の中で規格化されている形式ですが、べつに MPEG-4 の略ではなく MP4 が正式名称です。混乱しなくて助かりますね。MP4 は MPEG-4 で規格化されてるとはいえ、中身が MPEG-4 Video や AVC だとは限りません。HEVC とか MP3 とか普通に入ります。気をつけましょう。

MP4 は ISO base media file format (MPEG-4 の Part 12 で規格化) というフォーマットの派生で、同じ ISO base media file format から派生した 3gp などのフォーマットと少し内容が違う程度でほぼ同じ形式になっています。ISO base media file format は長くてわかりづらいので 3gp などもだいたい MP4 とひとくくりにされてることが多いようです。

ISO base media file format 自体も QuickTime の mov 形式から派生したフォーマットなので、とても息の長いフォーマットです。

ちなみに .3gp や .3g2 は携帯電話の規格を作っている団体によって作られたファーマットのようです。ケータイ電話の着信音とかいじってた人は良く見たんじゃないでしょうか。

内容としてはボックスと呼ばれる単位でデータが格納されます。ボックスは名前と長さと内容のデータで構成される単位で、ボックスの内容として他のボックスを入れて階層構造にすることもできます。またボックスの名前を見て未対応の物は無視するといったこともできるため、特殊なボックスを作って拡張することも可能です。

ただ、ボックスの長さが決まらないと書き込めないためライブストリーミングには向いていません。というか基本的にはそのままではできません。ボックスの長さはボックスの先頭にあるヘッダにつくのですが、動画のデータ本体の長さがどんどん伸びていくので長さが書き込めません。MPEG-DASH などストリーミングで MP4 を使う場合は、動画本体を含まないファイルと動画本体を細切れにしたファイルに分けて扱います。

コーデックはある程度決まっていますが、ファイルタイプという 4 文字で識別されているので再生できるコーデックさえ判別できればいくらでも増やせはします。ただやっぱり登

録されている以外のコーデックが再生できることはあまり期待しない方がいいでしょう。

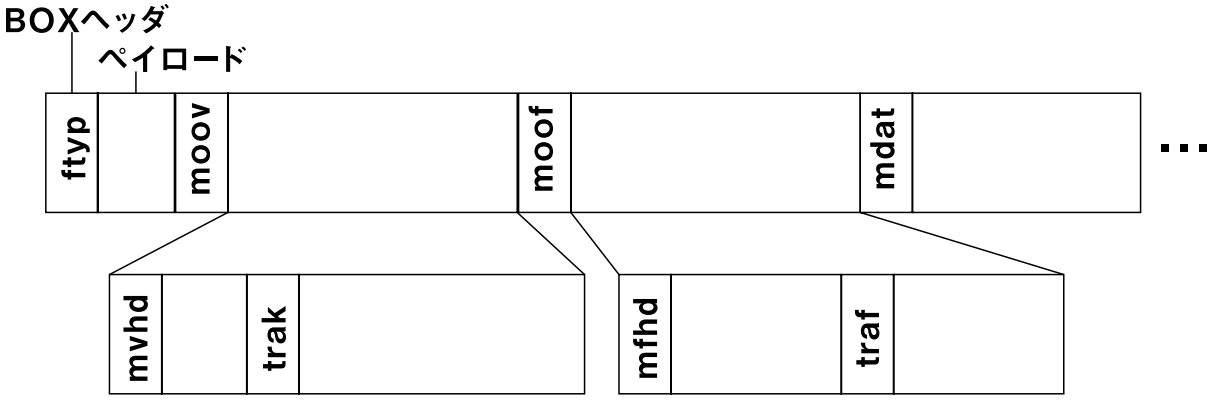


図 4. MP4 の構造

4.2. AVI (RIFF)

Windows で動画を扱うための Video For Windows 用に作られたコンテナ形式です。Video For Windows は Windows 3.1 の頃の話です。元になったのはフォーマット RIFF という形式で、WAV ファイルなども同じ RIFF 形式が元になっています。

中に入るコーデックが FOURCC と呼ばれる 4 文字で識別され、FOURCC が合うデコーダが見つかればあと中身はコーデック次第という簡単な作りになっているため、なんでも入れようとすれば入ります。

当然ながら Windows でよく使われましたが、さすがに古いこともあって今では ASF など他の形式にとってかわられました。単純なので今でも無理矢理使う人も居るようですが、ファイルサイズが 4GB を越えられないなど制限も多いので新しく採用するようなものではないでしょう。

4.3. Advanced Streaming Format / Windows Media Video (ASF/WMV)

最近まで Windows でよく使われたコンテナ形式です。

ファイルとして良く見るのは WMV や WMA だと思いますが、コンテナとしては ASF(Advanced Streaming Format) という形式です。ASF に WMV コーデックで映像が入ったものを特に拡張子 .wmv に、ASF に WMA コーデックで音声だけが入ったものを特に拡張子 .wma としているだけです。

ちょっと前まではよく使われていたのですが、最近では Microsoft がもう力を入れておらず、

公式の作成ツールすらほぼありません……。Microsoft は独自のコーデックやコンテナ形式をやめて MP4 など一般的な物を使うような方向になっているようです。

このコンテナ形式も AVI と同様に FOURCC でコーデックを識別してデコーダを探すのでけっこうなんでも入るんですが、Windows Media Player はストリーミング再生では WMV/WMA 以外を認識してくれなかったりと謎な動作をします。WMV/WMA 以外で使うのはあまり想定されてなさそうです。

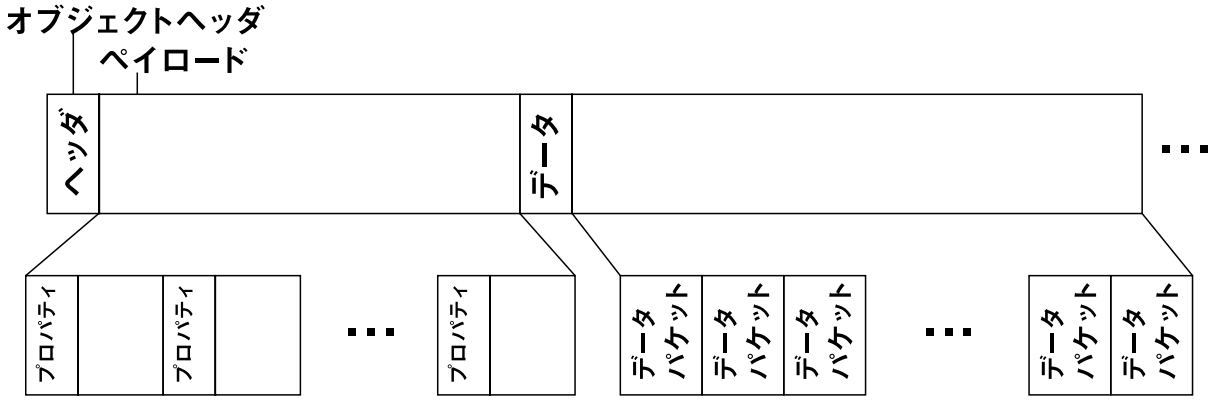


図 5. ASF の構造

4.4. MPEG-2 Transport Stream (MPEG-2 TS)

MPEG-2 の Part 1 で規格化されているコンテナ形式で、MPEG-2 PS とひとまとめに MPEG-2 System と呼ばれることもあります。

MPEG-2 で規格化されているものですが、中身が MPEG-2 Video や MP3 だとは限りません。MPEG-4 Video が入ったり AVC が入ったり HEVC が入ったりとしますので注意してください。コンテナとコーデックは分けて考えましょう。

ファイルとして直接見ることはあまりないと思われますが、実はいろんなところで使われています。

テレビのデジタル放送に使われているので、テレビを見る人は気付かないうちに使っています (MPEG-2 はもともとテレビ等の放送用に作られた規格です)。PC でデジタル放送の録画をする人は直接ファイルとして見てるかもしれません。

AVCHD や Blu-Ray のビデオでも使われています。これらは拡張されているので MPEG-2 TS そのままでは無いようですが……。

内容は 188 バイト単位のパケットで構成されています。小さな MPEG-2 TS ヘッダの後にデータがついて 188 バイトで一区切りになります。188 バイト毎に Sync バイトとい

う特定のデータが来るため、データの破損などでどこまで処理したかわからなくなっても、188 バイト毎に来る Sync バイトを見つけるとそこから処理の再開ができます。テレビの放送波などデータが破損しやすい用途を考えられて作られているのがわかります。

ただ 188 バイトで 1 パケットでは映像データなどを入れるには足らないので、複数のパケットで一つの PES パケットという音声や映像データを入れるパケットを構成することになります。

対応するコーデックは番号で決まっているのでなんでも入るとはいきません。ただ MPEG で使われるようなコーデックはすぐ追加されるので、対応するコーデック自体は多くあります。

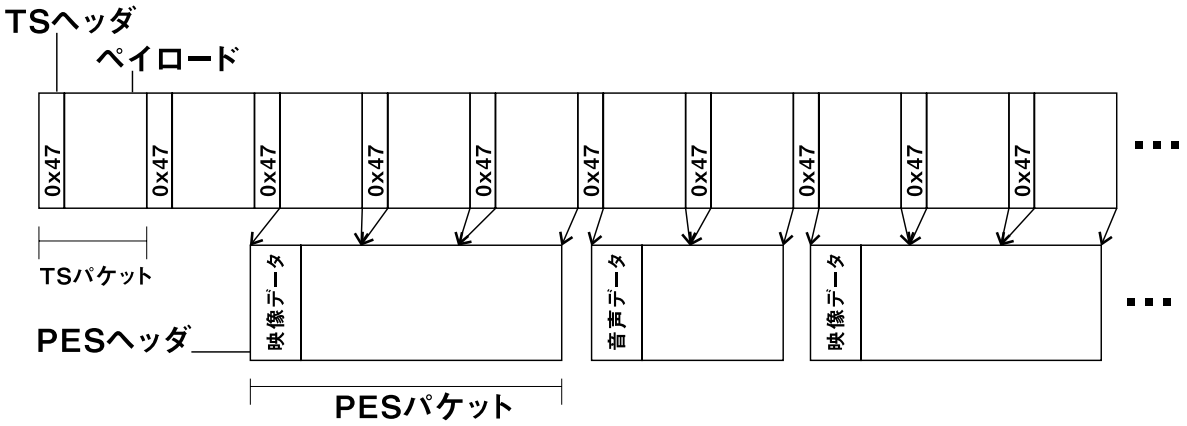


図 6. MPEG-2 TS の構造

4.5. MPEG-2 Program Stream (MPEG-2 PS)

MPEG-2 の Part 1 で規格化されているコンテナ形式で、MPEG-2 TS とひとまとめに MPEG-2 System と呼ばれることもあります。中身も MPEG-2 TS とだいたい似たようなものです。

これもファイルとして直接見ることはあまりないと思われます。

MPEG-2 TS がストリーミング用なのに対して MPEG-2 PS はディスクやファイルなど固定メディア用です。DVD の中身に .vob として使われています。

対応するコーデックは MPEG-2 TS と同じです。

4.6. Matroska / WebM

非営利団体によって作られているフリーでオープンなコンテナフォーマットです。

近代的でクロスプラットフォームなフォーマットを意図して作られてるようで、EBML と

いうバイナリ版 XML のような階層構造のある拡張性の高いフォーマットを使っています。ストリーミング再生も仕様上できなくはないですが、どちらかというとディスクに保存された状態からの再生がメインに考えられているようです。

WebM は Matroska をベースに Google が作ったフォーマットです。ヘッダが微妙に違うのと中に入れられるコーデックが決まっている程度で、ほぼ Matroska そのままです。WebM は Youtube など Google のサービスで使われているほか、Wikipedia に上がってるビデオにも WebM の物があります。

Windows 10 では標準で Matroska の再生できるようになりました。当然中のコーデックが再生できればの話ですが。

中身は先にも書いた通り、EBML という階層構造のあるフォーマットで構成されています。MP4 のボックス構造と似た感じになっています。MP4 と似ているということでストリーミングには向いてないんですが、仕様としては一応サポートしていて、動画データ本体のが入る部分のデータ長を -1 にすることで、不定長を表すことにしています。ただ、できるとはいえあまりストリーミングに向いた構造とは言えないので基本的にはファイル向けでしょう。

対応コーデックは、コーデック ID が文字列で入るためかなりなんでも入れようとすれば入ります。公式サイトには標準的なコーデック ID の表があります。FOURCC で AVI と同

様の指定もできるので AVI からの移行にも使われたりします。

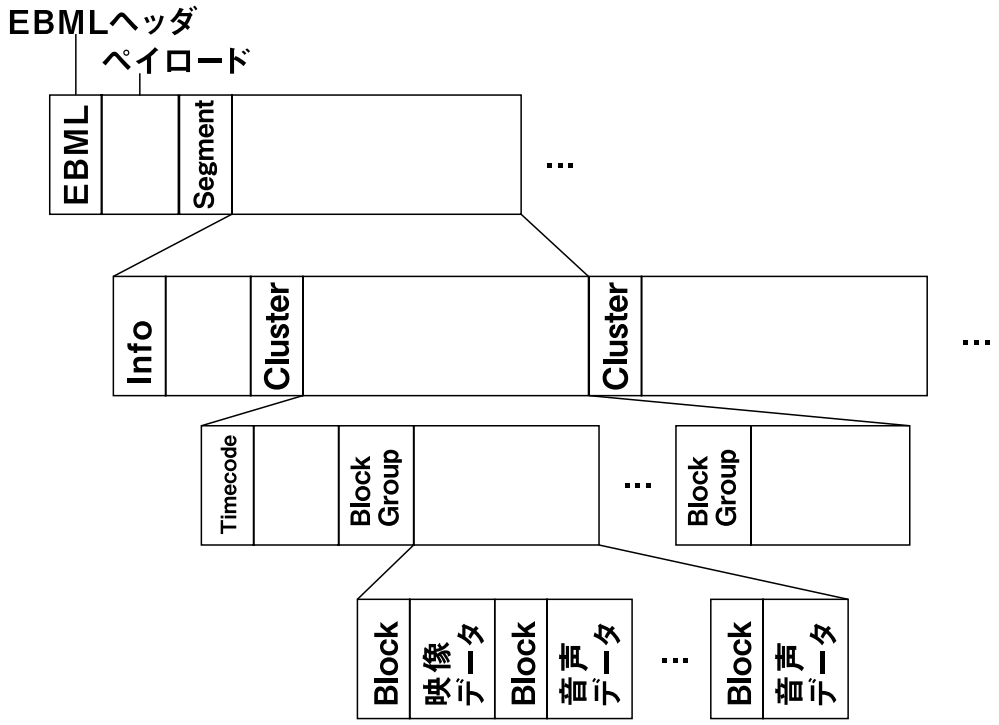


図 7. MKV の構造

4.7. Flash Video (FLV)

Flash で使われていたコンテナフォーマットです。今でも使えなくはないですが、今では Flash も .f4v という MP4 ベースのフォーマットを使うのを推奨しています。

視聴する際に直接ファイルとして見ることはあまりなかったかもしれませんが、一昔前の Web 上でのビデオ配信にはよく使われていました。今となっては Flash が廃れ気味ですし、Flash を使うにも MP4 が再生できるので積極的に使うことはないでしょう。

内容は Flash のストリーミングプロトコルである RTMP と近い構造の、FLV タグというのが並んだ形式になっています。各 FLV タグは映像や音声の 1 フレーム分になっており、FLV タグの前後にタグの長さが付くため、途中でデータ化けや欠落があっても復帰するのが容易になっています。あまりこれを UDP でやりとりすることはないと思いますが、まあストリーミング向きな構造です。また、RTMP で送受信するのに構造が似ているので、RTMP を介してやりとりするのには便利だったりします。

対応コーデックは独自の番号で決まっているので、決まった物しか入りません。さらにもう追加もされてないので新しいコーデックは使えません。最新のコーデックは映像で H.264、

音声で AAC です。ある程度の対応コーデックは RTMP の解説のところにあるのでそちらを参照してください。

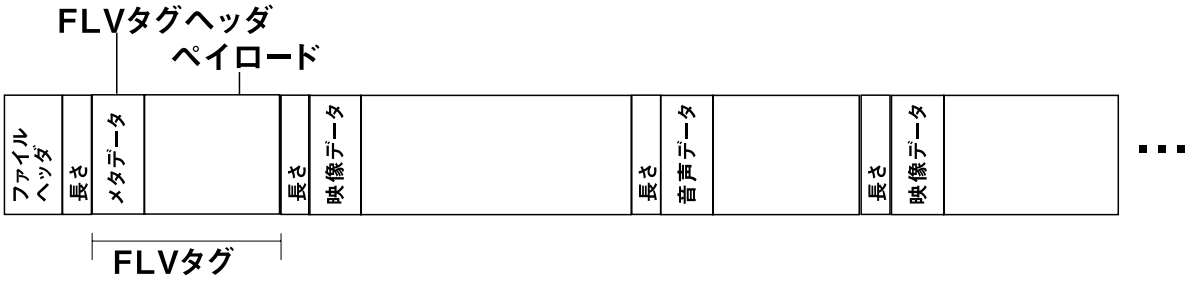


図 8. FLV の構造

4.8. Ogg

Xiph.org という非営利団体によって作られているフリーでオープンなコンテナフォーマットです。

同じオープンなコンテナである Matroska との違いですが、こちらの方がより単純でストリーミング向けのフォーマットになっています。

中に Vorbis というコーデックで音声を入れて Ogg/Vorbis と呼ぶのは聞いたことある人も居るんじゃないでしょうか。MP3 流行期かその後に一部で流行ったような気もします。

Wikipedia で見れるビデオは ogg に入っている物が多いです (Ogg/Theora か WebM のみ)。

内容は Ogg ページというパケットが沢山並んだ形式になっています。Ogg ページは必ず OggS という文字列で始まるので、データ化けや欠落が発生しても次に処理をするべきデータがすぐに見つけられるというストリーミング向きな構造になっています。ただし 1 ページ内に 1 フレーム分のデータが収まるとは限らず、デコードには複数のページを結合して扱う必要がある場合もあります。FLV と MPEG-2 TS の中間くらいの構造ですね。

コーデックは 8 バイトの識別子で指定するため、いくらでも入れられますが、コーデック側でどう Ogg に入れるかの規定が必要になるため、なんでも簡単に入るとは言い難いものです。実際に対応しているコーデックはフリーの物がメインなのでそんなに多くありま

せん。

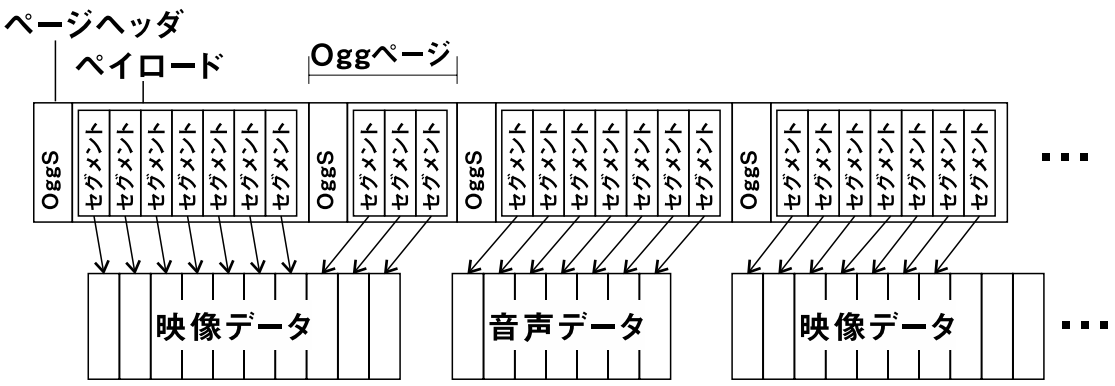


図 9. Ogg の構造

4.9. コンテナのまとめ

ここまでで紹介した各コンテナの特徴をまとめておきます。

コンテナ名	主な用途	対応コーデック数	特徴
MP4	ファイル向け	いろいろ入る	• そのままではストリーミングはできない • 拡張しやすい
AVI	ファイルのみ	いろいろ入る	• ストリーミングはできない • 単純 • 4GB を越えられない
ASF/WMV	ファイルおよびストリーミング	いろいろ入る？	• 比較的ストリーミング向き • WMV/WMA 以外のコーデックは期待できない
MPEG-2 TS	放送やストリーミング向け	そこそこ多め	• ストリーミングに強い • いろんなところで使われている
MPEG-2 PS	ディスク向け	そこそこ多め	• DVD などで使われている
FLV	ストリーミング向き	少なめ	• 単純 • 対応コーデックが限られる
Ogg	ファイルおよびストリーミング	少なめ	• 比較的単純 • オープンで特許問題がない • フリーなコーデックがメイン

コンテナ名	主な用途	対応コーデック数	特徴
MKV	主にファイル向け	いろいろ入る	• ストリーミングもできる • 拡張しやすい • オープンで特許問題がない

表 3. コンテナの比較

5. コーデック

コーデックとは映像や音声データなどの圧縮方法です。

コーデックもいろいろあります。映像と音声では圧縮方法は大抵違うので、映像用と音声用といった違いもありますし、以前の物より綺麗なままサイズが小さくできるといった分かりやすいものから、処理が軽いことを目標に作られたもの、他の特許にひっかからないように作られたのもあります。コーデックはコンテナ以上に多様です。

コンテナは通常いくつかのコーデックをサポートしていますので、同じコンテナに見えても中に入っている映像や音声データのコーデックが異なる場合があります。視聴するにはプレイヤーがコンテナだけでなく、コンテナ内データのコーデックも全て対応している必要があるので注意が必要です。

個々のコーデックの解説もしますが、あまりコーデックについては詳しくないため、簡単なものになるのをご了承ください。

5.1. 音声コーデック

5.1.1. PCM

インターネットの動画で使われることはあまり無いと思いますが、基本と言えば基本なので一応紹介しておきましょう。

一定時間毎に波形の振幅を数値化して並べるだけのフォーマットです。昨今の音声フォーマットとしては基本です。

CD なんかがこのフォーマットになっています。WAV ファイルもよほどひねくれてない限り PCM が入っているでしょう。

マイクやライン入力などから録音するとこのフォーマットで来ます。これ以降で紹介する音声コーデックの圧縮元は PCM なので、無圧縮の音声と言えば PCM といってほぼ間違いないでしょう。

5.1.2. MPEG-1/2 Audio Layer-III (MP3)

おなじみ MP3。MPEG-1 Audio Layer-III のことです。

MPEG-3 の略ではない、というのは有名な話かと思います。ちなみに MPEG-3 自体は作ってる途中で MPEG-2 に吸収されたようで、欠番です。

MPEG-1 Audio というのが MPEG-1 の音声コーデックに関する規格です。そのうちで複雑度によってレイヤーが I ～ III まであるのですが、一番複雑なレイヤー III が MP3 です。Layer-I や Layer-II は MP1 や MP2 とも呼ばれたりもしますが、どちらもあまり PC やネットでは見かけないので割愛します。

MPEG-2 にも MPEG-2 Audio という音声コーデックがあるのですが、基本的に MPEG-1 Audio と同じでいくつか制約が増えてるだけです。なので MP3 は MPEG-2 Audio にも含まれます。ややこしいですね。

MPEG-1/2 Audio と言えば MP3、MP3 と言えば音声コーデック、と覚えとけばだいたい大丈夫です。

PC 等で使う音声の圧縮コーデックとして爆発的に普及し、減りはしたものの今でも現役で使われています。PCM と比べて 1/4 ～ 1/20 くらい小さくなるとのことです。後発のコーデックの方が綺麗なまま小さくできるのですが、そこまで小さくする必要がなければ十分綺麗なのと圧倒的な普及率の高さで未だ現役です。

MP3 ファイルも見掛けると思いますが、これは MP3 で圧縮された音声データにちょっとしたヘッダをつけただけの物です。音声だけのファイルだとコンテナに入れて多重化するまでもないので、こういう構成になることがあります。

5.1.3. Advanced Audio Coding (AAC)

AAC でおなじみ Advanced Audio Coding です。直訳すると「進化した音声符号化」で、一体何が進化したなのかって話ですが、MP3 の次を目指して開発されたらしいので、MP3 に対して進化した音声コーデックなのでしょう。これも MP3 と同等以上に広く普及しています。

同じサイズであれば MP3 より綺麗になっています。つまり同じ綺麗さを保ったままサイズを小さくできます。

AAC LC(Low Complexity) というのが一番単純な AAC で、より効率の良い HE-AAC(High-efficiancy AAC) や HE-AACv2 などの拡張がいくらかあります。

MPEG-2 の Part 7 と MPEG-4 の Part 2 の両方で規格化されているので、放送からインターネットまで広い範囲で使われています。ちなみに MPEG-4 Audio の規格にはいろんな音声コーデックが入っているので、何か一つの音声コーデックを指して MPEG-4 Audio と呼ぶ

ことはありません。

5.1.4. Windows Media Audio (WMA)

Microsoft が作った音声コーデックです。バージョンがいくらかあるようですが、大きくは WMA と WMA Pro で分かれ、普及したのは WMA です。

MP3 普及期にはポータブルオーディオプレイヤー (MP3 プレイヤー) で WMA が再生できるのも多くありました。

動画では WMV と組み合わされることが多いですが、WMV も下火となった今となっては使うことはあまり無いでしょう。

音質的には WMA は MP3 と同じくらい、WMA Pro は HE-AAC のちょっと下くらいに着ける実力はあるようですがいかんせん対応機器などが少なく普及しませんでした。

5.1.5. Vorbis

Xiph.org によって作られた音声コーデックです。仕様はオープン、実装もオープンソースであり、特許問題が無いのが特徴です。

実力的にも少なくとも MP3 と同程度かそれ以上の品質があるようです。

オープンで特許フリーなこともあり Ogg と組み合わされて Ogg/Vorbis として、MP3 代替で使われたこともありました。また、PC ゲームなどでの採用も多くありました。

今でも十分使えますが、特許料無料でもっと効率の良いコーデックも出ているため用途は狭いでしょう。

5.1.6. Opus

比較的新しい音声コーデックです。比較的低ビットレートでのインタラクティブな音声のやりとりに向いています。つまりライブ配信やボイスチャット向けです。

Skype で開発されたボイスチャット用 (低ビットレート) の SILK というコーデックと、Xiph.org が作った Vorbis 後継の低遅延のコーデックである CELT を組み合わせたものです。どちらも低遅延なのが特徴で、ボイスチャットやライブ配信に向いてます。

リファレンス実装がオープンソースであり、既知の特許については無償で使えるのも嬉しい点です。

既にボイスチャットアプリではよく使われており、Web ブラウザ上でボイスチャット等を実現するための WebRTC では必須のコーデックとなっているため、使われるところは今後も広がっていくでしょう。

5.1.7. その他

可逆圧縮 (劣化無いやつ) のコーデックについては、ライブ配信ではあまり使うことはないでしょうから省略します。とりあえず名前だけ挙げときますので、気になったら調べてください。

- FLAC
- Apple Lossless
- MPEG-4 ALS
- などなど

5.2. 映像コーデック

5.2.1. Motion JPEG

JPEG 画像を並べただけのまさにパラパラマンガなコーデックです。

他の映像コーデックは画像毎の差分を取って小さくしたりするのですが、Motion JPEG では差分とか無しで単に画像を並べます。当然他に比べて効率は良くないのですが、単純なのでネットワークカメラなど高度な処理ができない機器で使われてたりします。

とはいえさすがに効率悪すぎなので、一般的なビデオ配信に使われることはありません。

5.2.2. MPEG-1 Video

MPEG-1 の Part 2 で規格化されている映像コーデックです。

ビデオ CD などあまりサイズが大きくない用途のために作られた映像コーデックです。え？ビデオ CD 知らない？ DVD より昔に CD にビデオを入れたのがあったんですよ。画質も悪くて流行らなかったですけど。とにかくだいぶ前の規格です。

ビデオ CD 自体は流行りませんでしたが、MPEG-1 Video 自体は比較的低ビットレート向けの規格なこともあり PC で使える動画コーデックとしてけっこう良く使われました。単に MPEG 形式の動画と言った場合には MPEG-1 Video だった時代があったのです。今は MPEG の規格もたくさんあるので単に MPEG と言うとよくわからなくなりますね。

5.2.3. MPEG-2 Video (H.262)

MPEG-2 の Part 2 で規格化されている映像コーデックです。MPEG-2 Video は ITU(国際電気通信連合) と共同開発のため、ITU の規格番号である H.262 ともたまに呼ばれます。

MPEG-1 Videoの拡張で、DVDビデオやデジタルテレビ放送などに現在でも使われています。

放送向けなのであまり低ビットレート向きは想定されてないようです。回線速度が上がった今のインターネットでは無理ではないでしょうが、より効率の良いコーデックもありますので、インターネットでの配信に使われることは少ないでしょう。

5.2.4. MPEG-4 Video (H.263)

MPEG-4 の Part 2 で規格化されている映像コーデックです。ITU-T で規格化されている H.263 を改良したもので、そのまま同じ物ではないですが一部互換性があります。

なお、MPEG-4 には MPEG-4 Video とは別に MPEG-4 AVC という映像コーデックがあるのでごっちゃにならないよう注意が必要です。

MPEG-4 自体が PC やインターネットでの利用を想定した規格のため、MPEG-4 Video も MPEG-2 より低ビットレートで使えることを目標として作られています。PC 周りでは当然のことながら多く採用されており、Flash では H.263 が使えたり、Windows では Microsoft の実装があり、後述の Windows Media Video や他のコーデックの元になったりとあちこちで使われています。

今となってはより高圧縮な MPEG-4 AVC があるため、そちらが使われることの方が多いでしょう。

5.2.5. MPEG-4 Advanced Video Coding (AVC, H.264)

MPEG-4 の Part 11 で追加で規格化された映像コーデックです。ITU(国際電気通信連合) と共同開発のため、ITU の規格番号である H.264 とも呼ばれることもよくあります。MPEG-4 を省略して単に AVC と呼ばれることも多いです。

MPEG-4 Video のところにも書きましたが、MPEG-4 AVC と MPEG-4 Video は、どちらも MPEG-4 規格内の映像コーデックでありながら別物です。めんどうなことに両方とも使われるから分かりづらいですが、まあ MPEG-4 AVC の方は単に AVC や H.264 と呼ばれることが多いので MPEG-4 Video との区別はなんとかつくでしょう。

2017 年現在ではほぼ主流になっているコーデックなので、一度くらいは聞いたことがあるでしょう。

MPEG-2 と比べて 2 倍以上圧縮効率を上げることができ (同じ画質ならサイズ半分以下)、低ビットレートで高圧縮な映像から高ビットレートで高画質な映像まで幅広くサポートしています。また、従来のコーデックより単純なハードウェアでデコードしやすい作りになっているため、小型で省電力なモバイル機器などでも使い易く広く普及しました。

デコードはともかくエンコード処理は効率が良いだけあって重くなりがちですが、機器の性能が上がったりハードウェアでのエンコードアクセラレーションを内蔵する機器が増えてきたため、あまり問題にならなくなっています。

インターネット関係で良く使われているコーデックではありますが、高ビットレートも対象としているためデジタルテレビ放送 (日本ではワンセグ) や Blu-Ray のビデオでも使われています。

2017 年現在のライブ配信ではほぼ H.264 一択と言って良いくらい普及しています。さらに効率の良いとされる次世代の映像コーデックも続々と登場していますが、そう簡単に移り変わりもしないのでまだしばらくは H.264 が使われ続けるでしょう。

5.2.6. MPEG-H High-Efficiency Video Coding (HEVC, H.265)

MPEG-H で規格化されてる映像コーデックなんですが、誰も MPEG-H とか知らないので単に略して HEVC と呼ばれます。

これまた ITU と共同開発のため、ITU の規格番号である H.265 とも呼ばれます。

4K や 8K といった高解像度の映像をターゲットにはしているのですが、低解像度や低ビットレートでも同様に効率が良いため、H.264 の後継として普及が期待されます。H.264 よりさらに 2 倍効率的 (同じ画質でサイズ半分) だそうです。

ただその分処理が重く、デコード (再生) はまだしもリアルタイムでのエンコード (圧縮) はどんな機器でもというわけには行きません。普及にはまだ機器の性能向上を待つ必要がありそうです。

ライブ配信でデファクトスタンダードになっている RTMP が H.265 に対応していないため、別なプロトコルを採用する必要があります。H.265 をライブ配信に使おうというにはまだ重いので、利用者が今気にしないといけない問題ではないのですが、開発者としてはそろそろプロトコルをどうするか考える時期にありそうです。

5.2.7. Windows Media Video (WMV)

Microsoft が作った映像コーデックです。MPEG-4 Video を改良したもののようです。

WMV には 7、8、9 と 9 Advanced Profile があり、9 以降に関しては標準化されて VC-1 という名前でも知られます。VC-1 は Blu-Ray のビデオでも使われるようです。

ちょっと前までは Windows で再生できるビデオとして、また、Windows Media Player や Silverlight へのストリーミング配信用のコーデックとしてよく使われていましたが、MS 自体が Windows Media Player や Silverlight や Windows Media Video に消極的になっているために、新しく採用されることは少ないでしょう。

拡張子 .wmv のファイルにはこの WMV コーデックの映像が入っています。その場合もコンテナとしては ASF なので注意が必要です。

5.2.8. VP6 (Flash Video)

今は亡き On2 Technorogies という会社が作った映像コーデックです。今は亡きと言いますがべつに On2 は潰れたわけではなく Google に買収されました。

Flash で使えるビデオコーデックとして採用され、Flash Video と呼ばれることもありました。昔 Flash でビデオ再生していた時は良く使われていたのではないでしょうか。あとから H.263 や H.264 が使えるようになったので結構昔の話になるでしょうが。

昔の FLV ファイルなどを引っ張り出してくると見られるかもしれませんが、今新しく使うことは無いでしょう。

5.2.9. Theora

Xiph.org が On2 VP3 を元に作った映像コーデックです。VP3 てのは上で紹介した VP6 より前のコーデックですが、Xiph.org に寄付されてオープンになりました。

相当古いコーデックを元にしてるだけあって、今となってはお世辞にも良いとは言えない物ですが、特許料フリーで使えるのが魅力です。そういうこともあってか Wikipedia に貼ってある動画でよく見かけます。

特許料やオープンソースなどの問題は後述の VP8 や VP9 で解決してしまってますので、今から新しく採用するようなものでもないでしょう。もし利点があるとすればせいぜい処理が軽いあたりでしょうか。

5.2.10. Daala

Xiph.org が Theora の後継として作っている映像コーデックです。

特許料フリーのオープンなコーデックですが、H.265 越えの高効率を目標として作られており、上手く実現できれば有望なものとなるでしょう。

一応動くものはありますが、まだ開発中ということで評価はまだまだこれからになりそうです。

5.2.11. VP8

Google が買い取った On2 Technorogies が作った映像コーデックです。WebM の中身です。

比較的新しいコーデックで H.264 とタメを張れる程度の性能がありますが、なによりエンコーダ / デコーダがオープンソースにされているだけでなく、特許料フリーで使えるのが魅力です。

Matroska コンテナに入れて WebM として出していますが、WebM には後継の VP9 が入っ

ている可能性があります。対応ブラウザで Youtube を見ると WebM を再生するようですが、もしかしたらコーデックは VP8 ではなく VP9 になってるかもしれません。

特許料無料などいい点はありますが、H.264 が流行りすぎた後から出てきたためにそんなに広くは使われている様子はありません。後継の VP9 に期待したいところ。

5.2.12. VP9

Google が作った VP8 後継の映像コーデックです。H.265 対抗らしいです。

VP8 と同様にエンコーダ / デコーダがオープンソースにされているだけでなく、特許料フリーで使えるのが魅力です。そのおかげもあってか現行のほとんどの Web ブラウザ (Safari を除く) でサポートされており、デコードのハードウェアアクセラレーションも多くの環境でサポートされています。

まだ流行り始めの H.265 対抗でありながら、対応環境も広いため期待が持たれます。ただ Apple の野郎はそっぽを向いてるようです。

5.2.13. AOMedia Video 1 (AV1)

Alliance for Open Media (AOMedia) という団体が作っている映像コーデックです。実質 VP9 の後継です。

VP9 を改良し、Daala や Thor の良いところを取り込んで H.265 を越えるコーデックを作っているようです。これも特許料フリーで使えます。

AOMedia 自体が Amazon, Cisco, Google, Intel, Microsoft, Mozilla, Netflix といったインターネットサービスやハードウェア関連の会社で作られた団体のため、VP9 と同様に Web ブラウザやハードウェアでの対応が早そうなことが期待されます。ただやっぱり Apple の野郎が入ってないのが不安なところですね。

また、リアルタイムビデオ (ライブ配信やビデオチャット) 用に低遅延も考えて作られているようなのでライブ配信向けとそても期待したいところです。

2017 年中には仕様が固まる予定とのことなので、広く使えるようになるのは 2019 年や 2020 年からになるかもしれません。

6. で、どのコーデックが一番強いの？

さて、いろんなコーデックを紹介してきましたが、どのコーデックが最強なんでしょうか。

用途が合っていれば新しめなコーデックの方が新しい技術が使われているので綺麗に小さくできることが多いのですが、じゃあ新しいコーデックさえ使っていればいいのかという

とそうでもありません。同じコーデックでもエンコーダ実装によって大きな差がついてしまいます。

コーデックは多くの場合、デコード方法について規定しています。こんなデータが入ってくるのでこういう風にデコードしてください、というのだけ決まっていて、逆にこんな風にデータをエンコードしてくださいとは決まっていません。というのも、再生さえできればどんな風にデータを作ってもかまわないからです。そのため高効率にできるコーデックでも、エンコード方法によっては低効率なデータになってしまうことがあります。

低効率なエンコーダが悪いのかというと一概にそうでもなく、リアルタイムエンコードでは効率を犠牲にしても処理速度を優先させる場合があります。一般的にはより綺麗にエンコードするにはより重い処理が必要になりますので、そこまで処理能力や時間が無い場合には品質を犠牲にするのは仕方ないことでしょう。

逆に理論的にはより低効率なコーデックでも、エンコーダの出来が良ければ高効率なはずのコーデックの物より綺麗で小さいデータが出来上がることもあります。

そんなわけで実際のエンコーダを評価してみないことにはコーデックの良さが計れないのが難しいところです。コーデック自体の良さは綺麗にできる理論値みたいなもので、その性能を引き出せるかどうかは実装にかかっています。ただエンコーダの評価といっても同じ程度のビットレートでどっちが綺麗か、同じ程度の綺麗さでどっちが速いかなどと、綺麗さという主観での評価をしなくてはいけないために比較もそう簡単ではないようです。

とりあえずハードウェアエンコーダは速い (軽い) けどとても効率が悪いというのは覚えていて損は無いでしょう。ソフトウェアエンコーダの方がたいていは綺麗になります。

ソフトウェアエンコーダの場合はいろいろと設定がいじれることも多く、同じエンコーダでも速度優先だったり画質優先だったりで効率がだいぶ変わってきます。ただいくつかのプリセットを選ぶ程度ならいいのですが、膨大なパラメータを設定できるエンコーダもあり、そういったものを上手く設定するのはなかなか難しいでしょう。

で、最終的にどれがいいかって話なんですが、みんな使ってるもの (人気のあるもの) の中から用途に合わせて選ぶのが簡単でいいんじゃないでしょうか。みんなが使ってるものならそう大きく外れることはないでしょう。情報を得やすいというのも利点です。

もうちょっと凝るなら、エンコーダの比較やパラメータ調整が趣味な人も (仕事な人も) 居ますので、そういった人がおすすめしたエンコーダやパラメータを使うのが良いでしょう。

もっと凝りたい人は趣味な人でしょうから、思う存分頑張ってください。

第4章
通信プロトコルの基礎知識

ここからは通信プロトコルについて解説します。

まずプロトコルとはなんぞやという話をすると、プロトコルとは通信手順のことです。通信にはデータの送る側と受け取る側が必要ですが、こっちとあっちが互いの言ってることを理解していないとデータのやりとりができません。じゃあこういう方法でデータをやりとりしようね、という取り決めがプロトコルです。

通信プロトコルでわかりづらい点としては階層構造になっているというところがあります。ある通信プロトコルの上に別なプロトコルが乗ってるのです。

簡単に例を挙げましょう。離れた場所の人と何か文章をやりとりしたいとします。

何語で書きましょうか？普通に日本語がいいでしょうかね。じゃあ日本語で書きます。これがプロトコルです。互いに日本語が読み書きできないとやりとりできませんしね。

次にどうやって送りましょうか。メール？郵便？ Twitter ？ IRC ？今時 IRC はないってか。まあとりあえず無難にメールにしておきましょう。これもプロトコルです。互いにメールの送受信ができないとやりとりできませんものね。

メールと日本語という二つのプロトコルを使うことにしましたが、これらは独立でありつつ階層化されています。メールは中身が日本語だろうが英語だろうがロシア語だろうが送れますので、メールと言語は独立しています。日本語はメールだろうが郵便だろうが IRC だろうが何を使っても送れるので、日本語と通信手段は独立しています。ただ日本語単体では送受信できません。相手がすぐそばに居れば日本語でやりとりできますけど、その場合も口頭でという通信手段を使っています。日本語は通信手段とは独立していますが、実際に誰かとやりとりする時には通信手段が別に必要になります。つまり言語でのやりとりは通信手段の上に成り立つものなのです。

さて、メールだので例を挙げましたが、動画の通信プロトコルに戻ってきましょう。メールとか日本語とかやりとりしたいわけじゃなくて、動画をやりとりしたいんだった。

なんでそんな階層構造とかいう話になったのかというと、説明する必要があったからですね。

ストリーミングのプロトコルということで、実際に動画を載っけるプロトコルを説明したいんですが、その前にもうちょい下のプロトコルを説明する必要があるのでした。

とはいえデータが光ケーブルに乗ってるのか無線で飛んでるのかとかいうところまで掘る

つもりはありません。ちょっと下までな。

1. TCP と UDP

今のインターネットでは主に TCP や UDP といったプロトコルでデータがやりとりされています。TCP も UDP も送れるデータ内容は全く同じなのですが、TCP の方が高機能です。

比較内容	TCP	UDP
データ内容	なんでも	なんでも
データ受信の順番	送られた順に並び換えて受け取る	順不動でとりあえず届いた順に受け取る
届かなかった場合	自動で送り直す	届いたかどうかすらわからない
沢山データを送る場合	送れるか様子見つつ少しずつ流す	送れるかどうか知らんけど全力で流す

表 4. TCP と UDP の比較

表のような関係になっています。なんだ TCP 最強じゃねぇか。UDP いらないわ。

……と、言いたくなりますし、実際ファイルのやりとりでは TCP がよく使われているのですが、ストリーミング配信ではそうとも言えなくなります。

TCP は表のような高機能なことを勝手にやってくれるのですが、必要なくても勝手にやってくれます。

たとえばデータの欠落です。インターネットは途中でいろんなところを経由していますので、データが上手く届く保証がありません。たまにぽろっとデータが届かないことがあります。TCP ではデータが届いたら届いたよって返事を貰うことにしており、そろそろ届いててもおかしくない頃になっても返事が来なかったらもう一回送っておきます。

さて、ライブ配信してる時にデータがぽろっと抜けたらどうなるでしょう。画面や音が途切れてしまいます。そりゃいけません。でもよく考えると次の瞬間には次のデータが届きますので、途切れるのは一瞬ですね。普通は大量にデータが抜けることはないので、1 秒に満たない一瞬が抜ける程度です。さすがにずっとデータが届かなくなったら回線が切れてる気がするので、それ以降途切れるのはどうしようもないですが、一瞬データが届かなかったくらいは気にしなくていいでしょう。まあ途切れたのが大事な一瞬ということもありますが、ちょうどよく大事な一瞬だけ抜ける確率は低いでしょう。

ですが TCP の場合は一瞬抜けたけどまあいいやなどということが分からないので勝手に送り直してくれます。結果として抜けた分のデータもあとから届きますが、次のデータが

届いて再生が進んでいれば、あとから届いた前のデータはもういらないデータだったのでした。

いらないデータを送り直してくるぐらいだったらまだいいのですが、TCP はデータ受信の順番保証もしてくれているので悪化します。動画 1 枚目・2 枚目・3 枚目が順番に送られてきた時に 2 枚目が届かなかったとしましょう。1 枚目と 3 枚目だけが届きました。この時アプリケーションからは 1 枚目だけが受け取れます。TCP レベルでは 3 枚目も受け取れてるんですが、2 枚目が先に来るはずなので順番保証のためアプリケーションには渡しません。しばらくすると送り直された 2 枚目が来るので、アプリケーションはしばらくしてから 2 枚目と 3 枚目が一気に受け取れます。2 枚目の画像がべつにどうでもよい画像であれば、どうでもいい 2 枚目来るのを待ってるより、いいから 3 枚目以降を再生してほしいところですが TCP はそれを許してくれません。

TCP にはもう一つ、輻輳制御という機能が付いています。データを一度に沢山送ろうとした時に、途中でデータが沢山すぎて送りきれないところがあるとデータはそこで捨てられてしまいます。TCP だと届かなければどうせあとで送り直すのですが、また沢山送ったらまたいっぱいになって捨てられてしまいます。このネットワークいっぱいでデータが捨てられる状態がネットワークの輻輳と呼ばれる状態です。TCP ではこれを回避するためにデータを沢山送ろうとしても、最初はちょっとずつゆっくりデータを流して大丈夫そうだったら一度に流すデータを増やしていく、だめそうだったら大丈夫そうな分だけゆっくり流す、という制御をしてくれます。でもストリーミング配信だとそんなまだるっこしいことやられるとちょっと困ります。送れるなら送る、だめならだめで諦める！どっちかにしろ！！

UDP だとどうでしょう。UDP では TCP のような高機能はないので、送られたデータが届いた順にアプリケーションで受け取れます。届かなかったら、そもそも届かなかったということも分かりません。動画の映像 1 枚目・2 枚目・3 枚目が順番に送られてきた時に 2 枚目が届かなかったとすると、UDP だとアプリケーションからは 1 枚目と 3 枚目が受け取れます。2 枚目は？知らん。届かなかったのかもしれないし遅れてあとから来るかもしれません。まあでも無いなら無いで仕方ないので 1 枚目をちょっと長めに表示してそれから 3 枚目を表示しましょう、ということができます。1 枚目表示してる間に上手いこと 2 枚目が来たら表示すればいいですしね。

こうなると UDP が有利そうですね！アプリケーションの選択肢が増えました。ただ選択肢が増えたということは面倒だということでもあります。TCP の場合は映像データを受け取った順に表示すれば 1 枚目・2 枚目・3 枚目……と表示できるのですが、UDP の場合は順不動で来るため画像一枚毎にこれは 1 枚目、これは 2 枚目で……と順番を別につけておかないとおかしな順番で表示してしまうことになりかねません。また、抜けちゃいけないデータがあった時に、届いてなさそうだからもう一回送って欲しい (もしくは送ろう)

といったことをアプリケーションが自分で制御しないといけません。

輻輳制御も必要になります。ネットワークが途中でいっぱいになって受け取れないのに延々と送り続けると、相手だけじゃなくて途中のネットワークを使っているみんなにも迷惑かかってしまいます。なんかデータが送れてなさそうとなったらデータ量を減らすなり、諦めてデータを送るのを止めるなりするのをアプリケーションがなんとか判断して制御しないといけません。こりゃ大変かも。

TCP と UDP どちらも一長一短あります。だからこそ両方使われているわけですが。一般的にファイル転送なんかには TCP が使われて、ストリーミング配信なんかには UDP が使われます。

といってもストリーミング配信でもいろいろあって TCP が使われることが多いんですけどね……。

1.1. UDP-Lite

UDP はストリーミング配信向きという話をしましたが、さらにストリーミング配信向きな UDP-Lite というプロトコルもあります。

通信中にデータが届かないというパターンについて解説しましたが、他にデータは届いたものの一部が化けてる (おかしくなっている) というパターンもありえます。この時は TCP も UDP もデータが壊れてるのを検出し、届かなかったものとして捨ててしまいますが。一方 UDP-Lite は捨てずにちょっとくらいおかしくてもいいことにしてアプリケーションに渡します。

届いた動画データの一部が壊れてたことを考えましょう。画像の一部の色がおかしくなっています。まあでも他の部分は大丈夫そうだし、表示されるのは一瞬だし、捨てるくらいだったら表示しちゃってもいいんじゃないでしょうか？というのを実現できるのが UDP-Lite です。

UDP-Lite は TCP や UDP に比べてだいぶ新しいプロトコルですし、対応 OS もまだ少ないため実際に広く使われていると聞いたことはありません。ただ動画のストリーミングは今後も増えこそはすれ減りもしないしょうから、UDP-Lite も今後徐々に使われるようになるかもしれません。

2. アプリケーション層のストリーミングプロトコル

TCP だの UDP だのといった下層 (トランスポート層) のプロトコルについてちょこっと説明しましたが、実際にアプリが直接実装する、アプリケーション層のプロトコルについ

て解説しましょう。

2.1. Hyper Text Transfer Protocol (HTTP) でのプログレッシブダウンロード

みなさんご存知 HTTP です。さすがに聞いてことないって人は居ないでしょう。でも HTTP てストリーミングのプロトコルではないのでは……？ Web ページだのファイルだのダウンロードするアレでしょ？

その通りなんですが、ダウンロードしつつダウンロードできたところまで再生しちゃうという方法で原始的なストリーミング再生ができてしまいます。このダウンロードしつつ再生するのをプログレッシブダウンロードと呼びます。

Web ページに Flash でプレイヤーを埋め込んで FLV を再生してた時代には普通だったのですが、いまどきはそういうサイトもほとんど無いんじゃないでしょうか。外の動画置けるサイトに上げちゃいますしね。

原始的な方法だけに特に難しいことなく実現できてしまうのが特徴です。普通にファイルを HTTP サーバで配信するだけです。クライアント側は落としたところまで再生できるような作りになっている必要がありますが、FLV や MPEG-2 TS なんかだと多くの動画プレイヤーで対応しています。MP4 なんかはそういう作りになっていないため、通常はプログレッシブダウンロードでの再生はできません。

ライブ配信の場合は事前にファイルの長さが決まりませんが、HTTP ヘッダの Content-Length を出力しないことで実現できます。通常の HTTP サーバそのままではできませんが、Web アプリや CGI を作れば簡単に実装できます。

普通の HTTP でのファイルダウンロードなので、制限がかかることがあまり無いという利点もあります。ネットワークによってはセキュリティのためだのなんだので余計なポートへの接続や変なプロトコルでの通信が許可されない場合がありますが、さすがに HTTP でのファイルダウンロードが止められる環境はあまりないので、少々セキュリティに厳しいネットワーク環境でも視聴できることが期待できます。また、HTTPS を使うことで暗号化や改竄検知などを簡単に実現できます。

簡単で良さそうなプロトコルではありますが、HTTP だと標準ではバイト指定でしかシークができない、回線速度に合わせたビットレートでの配信 (複数ビットレートでの配信切り替え) ができないといった物足りないところがあります。バイト単位でしかシークできないのはライブ配信ではあまり問題にならないかもしれませんが、同時に録画も提供したり追いかけ再生ができるようにするなら欲しいですね。

いずれも独自もパラメータを Web サーバに渡して対応することはできるのですが、それだとサーバ毎にプレイヤーが必要になるので汎用性がありません。そこで後述の HTTP

Live Streaming といった HTTP 上でストリーミング再生をするためのプロトコルが作られました。

対応コンテナ

途中まででも再生できるものならなんでもかまいません。 再生するプレイヤーによります。

対応コーデック

対応のコンテナに入ってプレイヤーが再生さえできればかまいません。

2.2. HTTP Live Streaming (HLS) や MPEG-DASH など

HTTP Live Streaming (HLS) は HTTP でストリーミング配信を行うために Apple が開発したプロトコルです。

中身はとても簡単で、プレイリストファイルと数秒単位の細切れの動画ファイルで構成されます。

プレイリストファイルには細切れの動画ファイルの URL と各ファイルの長さ (秒数) が並んでいます。好きなファイルから順番に連続で再生させれば、数秒単位にはなりますがだいたい好きなところから再生できますね。素晴らしい。

プレイリストにはすぐ再生できる分の動画ファイルしか載せられませんが、ライブ配信の場合にはどんどん増えていくはずですがどうすればいいでしょう。プレイリストをリロードすればいいですね。定期的にプレイリストを読み込み直して新しい動画ファイルが増えてたらそれを再生するだけです。

複数ビットレートでの配信から回線速度に合わせた物を選ぶのはどうしたらいいでしょうか。プレイリストには動画ファイルだけでなく他のプレイリストの URL も載せられます。これは 300kbps 用のプレイリスト、これは 1Mbps 用のプレイリスト……といった形でビットレート毎のプレイリストを載せといて、自分の回線速度に合わせた物を選んで、中の動画を再生していくだけです。

この方法は実に簡単で、下手するとプログレッシブダウンロードより簡単です。プログレッシブダウンロードでは落としきってないファイルを途中まででも無理矢理再生できるプレイヤーが必要ですが、HTTP Live Streaming では数秒だけの動画ファイルを落としきってから再生すればいいだけです。あとはプレイリストを上から順番に再生していくだけ。簡単だね！

問題があるとすればオーバーヘッドがでかいところ、遅延が避けられないところでしょうか。数秒単位で動画ファイルとプレイリストを HTTP でダウンロードしないといけないので、何度も HTTP のやりとりが発生しています。HTTP もそんなに効率の良いプロトコルでは

ないので余計なデータ転送が発生していまいます。また、それぞれの区間は完全にダウンロードが終わってから再生するため、数秒間のファイルが生成できるまでダウンロードできません。このため区間ごとの秒数分だけは必ず遅延していまいます。1 秒毎のファイルを作るのもできなくはないのですが、毎秒ファイルのダウンロードリクエストがかかるためサーバの負荷としては厳しくなるでしょう。かといって 30 秒毎くらいにすると倍で 1 分くらいの遅延が発生します。ちょっとリアルタイムのライブ配信では不利な感じですね。

いくらか問題はあるものの、サーバは既存の HTTP サーバ (たいてい速い！) で良く、細かい動画ファイルをどんどん追加していってプレイリストを更新していくだけと実装もとても簡単です。再生する側も落としたファイルを順番に再生していくだけです。さらに Apple が作ったこともあり Flash が使えない iPhone でも視聴できるとなると Web ベースのストリーミング配信では勢力を拡大しました。

初期の HTTP Live Streaming は m3u プレイリストと MPEG-2 TS の動画ファイルだったのですが、あとから動画ファイルとして MP4 を細ぎれにできるように細工したやつも使えるようになりました。

HTTP ベースのライブストリーミングはみんな HTTP Live Streaming になるのかと思いきや、こんな簡単な規格なのでみんなして似てるけど別物をどんどんと作り始めてしまいます。

Microsoft はその頃 Silverlight を売っていこうとしていたので、Silverlight と自分のところの Web サーバである IIS で使える Smooth Streaming というものを作ります。 これはプレイリストの部分が XML で動画ファイルが MP4 です。Flash を作っていた Adobe も HTTP Dynamic Streaming という同じようなのを作ります。これもプレイリストの部分が XML で動画ファイルが MP4 です。でも別物なので互換性はありません。

そんな感じで似てるけど違うものが乱立してしまったので、統一した仕様が作られます。それが MPEG-DASH というものです。DASH は Dynamic Adaptive Streaming over HTTP の略だそうな。MPEG-DASH はプレイリスト部分は XML で動画ファイル部分は基本的に MPEG-2 TS か MP4 です。他の動画コンテナも使えるのでゆるゆるですが。

また同じようなものを……と言いたくもなりますが、拡張性もありますし国際規格なのでこれ以上似たような物が作られることはないでしょう。現在では HLS と MPEG-DASH でだいたい落ち着いています。Smooth Streaming や HTTP Dynamic Streaming は Silverlight や Flash とともに死滅するでしょう。

HLS の対応コンテナ

- MPEG-2 TS
- Fragmented MP4 (細切れ用に変形した MP4)

HLS の対応コーデック

具体的な規定はないですが、MPEG-2 TS や MP4 に入るものが想定されているようです。

MPEG-DASH の対応コンテナ

- MPEG-2 TS
- Fragmented MP4 (細切れ用に変形した MP4)

その他再生側が対応していればなんでも良さそうです。

MPEG-DASH の対応コーデック

コーデックについて具体的な規定は無いようです。

2.3. Windows Media HTTP Streaming Protocol

Windows Media Audio(WMA) や Windows Media Video(WMV) に特化した HTTP ベースのストリーミングプロトコルです。Windows Media というだけあって Microsoft が作ったやつですが、HLS や MPEG-DASH の解説で出てきた Smooth Streaming とはまた別で、もっと前に使われていたものです。

WMA や WMV 配信に特化したプロトコルということで、このプロトコルは ASF 形式のストリーミング専用です。ASF は Advanced Streaming Format の略なのでストリーミング用なんだろうなぁと予想が付くと思いますが、このプロトコルで使われることを想定したコンテナフォーマットなんですね。

DRM によるコピー制御が効くこともあって一昔前のビデオ配信では度々使われており、Windows Media Player や Web ページに埋め込まれた Windows Media Player、Silverlight で作られたプレイヤーで見てた人もそこそこ居るんじゃないでしょうか。Silverlight が出てくる前は Windows 専用だったので見れる環境は限られたのですが、そもそも PC といえばほぼ Windows だしケータイも動画が視聴できるような回線でなかった時代なのでそれほど問題にはされてなかったようです。

基本的にクライアントはサーバに HTTP の GET や POST に特殊な HTTP ヘッダを付けて命令を発行し、サーバがそれに応じて状態を変更したり動画データを送ってきます。サーバ側からの通知は動画データに、通知用の特殊なデータを混ぜ込むことで実現します。

プレイリストをサーバ側で管理するサーバサイドプレイリストという機構も持っており、クライアントは次のプレイリストを再生といった命令をサーバに送ることで、サーバ送ってくる動画をが勝手に切り替えてくれたりといったことができます。クライアントよりサーバ側でなるべく処理をする思想のようです。

サーバが状態を持っているので普通の HTTP に比べるとだいぶ複雑になりますし、専用のサーバが必要になります。基本的には Windows Media Server という専用のサーバを使っ

て配信します。ただ Windows Media Encoder や Microsoft Expression Encoder といった動画エンコーダは、このプロトコルでライブ配信ができるサーバになることができたため、個人レベルでは直接 Windows Media Encoder に Windows Media Player を接続してライブ配信するというスタイルがライブ配信の黎明期には行われていました。エンコーダへの直接接続では大規模な配信は無理でしたが、黎明期ですので数十人程度もつなげれば十分でしたし、独自にリレーサーバを作って分散させたりといった対処がされました。2017 年現在も一部ではまだ行われていますが、いずれのエンコーダも開発終了して久しく、MS は Windows Media 系自体からほぼ手を引いた状態ですので今後の発展は望めません。

対応コンテナ

- ASF

対応コーデック

- WMV(VC-1 を含む)
- WMA

2.4. Realtime Messaging Protocol (RTMP)

Adobe Flash や AIR の通信プロトコルです。Realtime Messaging というだけあってストリーミング配信だけでなく、いろんなデータをやりとりするのに使えるのが特徴です。

TCP ベースで通信は行われますが、HTTP とは関係ありません。当然ながら RTMP 用のサーバが必要になります。Adobe 製の Adobe Media Server の他に Wowza といったサードパーティ製のサーバや、Red5 といったオープンソースのものもあります。nginx は基本的に HTTP サーバでありながら RTMP サーバにもなるようです。仕様の公開はされているのですが、Adobe 独自のプロトコルであり、標準化されてるわけでもないのに沢山の実装があるのは珍しいですね。Flash がいかに強かったかを感じさせます。

ちなみに仕様は公開されているものの、仕様書の出来が大変悪く雑に省略されている部分があったりと、仕様書だけでの実装は難しく、一部意図して記載されてない仕様すらある始末です。読んでもわからない仕様書とかほんとひでえ。

このプロトコルはビデオやオーディオのやりとりだけでなく、Flash の ActionScript からオブジェクトデータを送受信することができます。やりとりするオブジェクトデータは特に規定はなく、ActionScript で自由に処理できます。サーバからクライアントへの送信だけでなく、一つの接続でクライアントからサーバにも双方向でのデータのやりとりができます。

これらの特徴により、ただの動画のストリーミング再生だけでなく、双方向でのビデオ会議やオンラインゲームなども実現できます。Flash ベースのブラウザゲームで多人数で遊んだ記憶がある人は実は RTMP にお世話になっていたかもしれません。

動画データの暗号化によるコピー制御もできるので有料配信にも使われていましたが、正当なクライアントと同様の動作で簡単に抜けてしまうようになってしまいました。Adobe としては今となってはビデオ配信に関しては RTMP ではなく前述の HTTP Dynamic Streaming などをつかってほしいようで、RTMP には H.264 の映像と AAC の音声を載せることができますがこれを H.265 対応などに拡張する気は無さそうです。

Flash が下火であり Adobe も力を入れてはなさそうな感じなので、今となってはあまり使うことはないでしょう……。

と思いきや、ライブ配信ではクライアントからサーバへの通信においてデファクトスタンダードと言っていい程使われているプロトコルになってしまっています。

RTMP は双方向通信ができ、動画データもクライアントからサーバへの方向に送ることができます。さらに Flash Player にエンコード機能自体が付いていたため、Web カメラとマイクからキャプチャしてエンコードし、そのままサーバに送信してライブ配信するというのが手軽にできました。さすがに Flash Player から直接ライブ配信というのは今はあまり使われないと思いますが、黎明期のライブ配信サイトがその形態で行われており RTMP を使った配信ができたため、外部のエンコーダソフトやハードウェアも RTMP での配信に対応しました。そして視聴に Flash を使わなくなった昨今のライブ配信サイトでも、配信側は既存の配信ツールが使えるようにクライアントからサーバへの動画送信部分に RTMP を使うことがほぼ標準になっています。Flash は使わないのに独自のプロトコルだけが使われている面白い事例ですね。

そんなわけで 2017 年時点では RTMP がクライアントからサーバへの動画送信に多く使われている状況ですが、RTMP に載せられる最新のコーデックが H.264 と AAC までです。どちらも効率と負荷を考えるとまだ十分現役でやっていけるコーデックなので足りてはいますが、RTMP の今後の拡張見込みがないため H.265 や Opus といった既に使える高効率なコーデックが今後も使えなさそうです。すぐに廃れそうという程ではないにせよ、近い将来には別のプロトコルに取ってかわられることが予想されます。とはいえ取って替わる別のプロトコルというのがなんなのかまだはっきりしていないのですが……。

対応コンテナ

プロトコル上で直接映像や音声を転送するためコンテナを必要としません。 ただし流すデータは FLV の中身とほぼ同等です。

対応コーデック

- H.263
- H.264
- FLV
- VP6

- AAC
- MP3
- Speex

などなど。

2.5. Realtime Media Flow Protocol (RTMFP)

RTMP ついでに RTMFP というプロトコルも紹介しておきます。

これも RTMP と同様に Adobe Flash で使われている通信プロトコルです。RTMP より後に開発され、RTMP の強化版といった仕様です。

RTMP とは違って UDP ベースのプロトコルで、TCP より遅延に対して有利です。データが届くことは保証されていませんが、大事なデータは RTMFP レベルで再送などを行って到達保証をすることもできるようになっています。

UDP ですので TCP より NAT を越えやすく P2P での通信がやりやすくなっています。そのためクライアント同士を P2P で接続し、データを分散して配信しあうアプリケーションレベルマルチキャストという機能を実現しています。サーバでの帯域を節約しつつ沢山の視聴者にデータを届ける仕組みです。

RTMP の後継としてはだいぶ良くできたプロトコルですが、Flash でしか使えず、RTMP のように Flash 以外で普及することが無かったため、今となってはもうあまり使われないものとなっています。RTMP と違って仕様もわかりやすくオープンになっているのですが、それより用途の方が普及には重要なようです。

対応コンテナ

プロトコル上で直接映像や音声を転送するためコンテナを必要としません。

対応コーデック

RTMP に同じ。

2.6. Real Time Streaming Protocol (RTSP) Realtime Transport Protocol (RTP)

RTSP と RTP は全然別物なのですが、いっしょに使われることが多いのでいっしょに書いてしまいます。

RTP は映像や音声をリアルタイム送信するためのプロトコルです。ストリーミング配信以外にもテレビ会議や IP 電話でも使われています。

RTSP は RTP でやりとりするまでの制御をするためのプロトコルで、これは主にストリーミング配信用のものです。

まず RTP についてですが、UDP ベースで映像や音声をリアルタイム送信するために作られたプロトコルです。UDP ベースなので途中でデータが消えたりしますが、リアルタイム送信を目的としてるのでちょっとやそっと消えたところで気にしません。

RTP は映像と音声を別に扱います。一つの接続で (UDP なので接続というほどしっかりつながりませんが) は映像と音声のどっちかしか送信できません。映像と音声の両方を使う場合には映像と音声を別々に送り、視聴側で合わせて再生します。

RTP 自体では細かい制御もほぼありません。基本的にはなんらかの方法で接続できた人にひたすらデータを送りつけるだけです。なんらかの方法で接続するってところがまず必要になりますが、そこは RTP の範囲ではないため他の接続制御用のプロトコルと組み合わせて使います。テレビ会議や IP 電話の接続制御には SIP といったプロトコルが使われますが、メディアサーバからのストリーミング配信の接続制御に使われるのが RTSP です。

RTSP 自体は映像や音声のデータ本体は扱いません。RTSP サーバにこの映像が見たい、この音声を聴きたいといったリクエストをすると、RTP で映像や音声を送り始めるといった制御をします。RTSP 自体は RTP と独立してるので、RTSP で接続制御をしておきながら RTP とは別なプロトコルで映像や音声をやりとりすることはできるのですが通常 RTP が使われますので、単に RTSP と言えば動画本体の転送はだいたい RTP だと思って間違いありません。

RTSP は結構古めのプロトコルでそんなに難しくもないことから、多くのメディアサーバでストリーミング配信用のプロトコルとして実装されています。古くは RealPlayer でのストリーミング配信にも使われていましたし、Windows Media Server も HTTP ベースの Windows Media HTTP Streaming Protocol がありつつも RTSP での WMV ストリーミングも使えるようになっています。Apple の Quicktime Streaming Server も対応していたため、Quicktime プレイヤーでストリーミング配信を見ることもできました。ただしコーデックの対応はまちまちなため、RTSP/RTP に対応してるといってもプレイヤー側で再生できるとは限らないのが難しいところです。

RTSP が今インターネットで使われるプロトコルかというとあまりそんなことはなさそうですが、RTP に関してはいろんな用途に使われており、またいろんなコーデックが載せられるため今後もどんどん使われていくでしょう。それに便乗して、とりあえず簡単に RTP で配信する方法としての RTSP というのはほそぼそと使われる可能性がありそうです。

対応コンテナ

プロトコル上で直接映像や音声を転送するためコンテナを必要としません。

対応コーデック

RTP 自体に規定はありません。 各コーデック毎に RTP に載せるための仕様が別途必要ですが、多くのコーデックが RTP に載せる仕様を規定しています。

2.7. WebRTC / ORTC

WebRTC は Google が開発した Web ブラウザでビデオチャットなどを実現するための機能です。作ったのは Google ですが開発はオープンに行われており、Firefox や MS Edge でも使えるようになっています。ORTC は WebRTC の拡張みたいなものでだいたい同じっぽいです。

WebRTC は機能セットのことでプロトコルそのもののことではないのですが、WebRTC の通信周りでは UDP ベースでの動画の送受信と動画以外のデータの送受信が行えます。Flash 無しで RTMFP と同じようなことができるようにしたかったんじゃないでしょうか。

RTMFP と違うのは既存のプロトコルを組み合わせて作られていることです。動画の送受信には RTP を暗号化できるようにした SRTP(基本的には RTP) が使われます。データの送受信には UDP 上で再送や輻輳制御を行えるようにした SCTP というプロトコルが使われます。通信開始時にはどんなデータを送るかを SDP というフォーマットでやりとりします。SDP は RTSP でも使われるフォーマットですね。この SDP でフォーマットをやりとりするのはちょっとダサいということで、その辺を改良したのが ORTC ということのようです。

これらの通信を Web ブラウザから行えるようにするのが WebRTC です。もちろんユーザーが直接いじるわけではないので JavaScript からの制御になりますが。

Flash 無しで Flash みたいなことをできるとあって注目の WebRTC ですが、まだ絶賛開発中ということでどんどん機能が追加されたり細かいところで変わっていったり各実装の足並みが揃ってない状態のようです。

対応コンテナ

RTP と同じなのでコンテナは必要ないですが、コーデック毎に RTP に載せる仕様が必要です。

対応コーデック

- VP8
- H.264
- Opus

などなど。 RTP に載るコーデックならなんでもいけるはずですが、いくつかが対応必須コーデックとして決まっています。

第5章 実用ライブ配信

ここからはライブ配信で実際に現在使われているものと、今後使われそうなものを簡単に紹介しましょう。

実際内部で何が行われているかは詳しく調べるのが難しいため、外からちょっと見てわかる程度の情報になってしまいますがご了承ください。

1. ライブ配信サイト

ライブ配信サイトはいろいろありますが、有名なのは次ようなものでしょうか。

- Youtube
- Twitch
- ニコニコ動画
- Ustream

配信者がこれらのサイトで配信しようとすると基本的に RTMP で H.264 と AAC の動画を送ることになります。ライブ配信では RTMP 以外の選択肢が現時点で無いため、H.265 などを使えるサイトは見かけません。

送った動画はそのまま視聴者には再配信されるわけではなく、サーバ側で再エンコードされ、複数のビットレートの動画が出来上がります。ただしニコニコ動画はライブ配信ではサーバ側での再エンコードは行われておらず、複数ビットレートの切り替えには対応していないようです。サーバ側でのリアルタイムな再エンコードは相当なパワーが必要になりますので、かなりお金のあるサイトでないと難しいのでしょう。

視聴はブラウザで見るには HLS や MPEG-DASH が使われます。

2. 今後使われそうなもの

ライブ配信で今後使われそうなものを挙げておきます。

2.1. RTMP

今後というか今使われているものですが、もう寿命だろと思いつつもまだしばらくは使われるんじゃないでしょうか。

確かに H.264 と AAC までしか載せられないのですが、固定回線での配信であれば十分なビットレートを確保できるためそんなに問題はありません。サーバ側で再エンコードする

なら配信側で高ビットレートの動画を送っても、視聴側は低ビットレートの動画を洗濯できますからね。

ただモバイル回線を利用しての配信も増えていますし、今後配信の解像度が上がっていくことも容易に想像できるため、いつまでも RTMP が使われることはないでしょう。

2.2. MPEG-DASH

これも今既に使われているものですが、Web ブラウザに限らずいろんなアプリでも視聴できるようになっているので今後も長く使われそうです。HLS も既に使われていて Apple が使い続けるでしょうからこれはこれで生き残るでしょう。

HTTP ベースでの配信なのでライブ配信には遅延の面でちょっと不利ではあるのですが、遅延をどうしても減らしたいという場面は現実的にはそれほどないために許容されそうです。というかサーバ側で再エンコードなどするとその時点で遅延が避けられませんしね。

2.3. WebRTC

WebRTC は UDP ベースでライブ配信にとって有望な機能かと思いますが、どの程度使われるようになるかは未知数です。

Web ブラウザから直接配信したいという場合にはもちろん使われると思いますが、配信用の機器に載せるのにはちょっと複雑というところがあり、RTMP の代替になるかはわからないところです。まだ H.264 の次世代クラスのコーデックが標準で使えるようにはなっていない点も代替としづらい理由でしょう。

サーバからの視聴には使われるようになるかもしれません。RTP なので遅延が少なくライブ配信に有利です。特に通信状況が安定するとは限らない無線通信が主になっている昨今では強いでしょう。ただそこまで遅延は気にならないし MPEG-DASH でいいじゃんと言われてしまえばそれまでです。動画配信サイトでもライブ配信に限って使われるといったことになるかもしれません。

3. 次世代コーデック

H.264 全盛の今ですが、次世代のコーデックは何が使われるようになるでしょうか。

H.265 はテレビの 4K や 8K 放送に使われるようなので、ハードウェアでのアクセラレーションについて発展が望めます。ただし H.264 と比べて特許料がかなり高くなっているようなので、そのあたりが多少普及のネックになることも考えられます。とはいえ少なくとも H.265 の再生ができないという機器はほとんど無くなるであろうことは予想されます。

VP9 は特許料無料ですし、既にデコードだけでなくエンコードのハードウェアアクセラ

レーションがいろいろな機器に搭載されています。ただ問題は Apple が採用しないため iOS や macOS で標準では再生できません。アプリで勝手にデコードすればいいのですがハードウェアアクセラレーションは効かないので電池食いますし、ブラウザでの再生は現実的に無理でしょう。AV1 も VP9 の後継として非常に期待が持てますが、やっぱり Apple が関与していないので同様にそっぽを向く可能性が考えられます。

個人的には特許料のかからないコーデックを応援したいのですが、現実的には Apple の抵抗により H.265 が強くなりそうですね。

第6章 参考文献

1. MPEG のサイト

MPEG の各規格についての概要が載っています。

MPEG のサイト

[http://mpeg.chiariglione.org/]

実際の規格の中身は ISO/IEC の規格書を見る必要があります。

2. MPEG の規格書

ISO や IEC のサイトから買うことができます。お高い。

ISO のサイト

[https://www.iso.org/ics/35.040.40/x/]

IEC のサイト

[http://www.iec.ch/dyn/www/f?p=103:22:876134264936::::FSP_ORG_ID,FSP_LANG_ID:3403,25]

ISO/IEC-13818 が MPEG-2、ISO/IEC-14496 が MPEG-4 です。一つのパートごとに 2 万円以上するので個人で買うのはちょっと厳しいものがあります。国会図書館で見たりコピーしたりできますので活用しましょう。

MPEG-DASH の規格書についてはなんでか無料で落とせます。やったね。

MPEG-DASH の規格書

[http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip]

3. ITU のサイト

規格書が落とせたり買えたりします。

ISO/IEC と共同開発になってるようなのは有料でも、一つ前のバージョンは無料で落とせたりするので活用しましょう。

ITU のサイト

[https://www.itu.int/en/ITU-T/publications/Pages/default.aspx]

H.222.0 (MPEG-2 System)

[https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11655&lang=en]

H.264 (AVC)

[https://www.itu.int/ITU-T/recommendations/rec.aspx?id=12904&lang=en]

H.265 (HEVC)

[https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12455&lang=en]

4. Adobe Flash の仕様書

Adobe のサイトから無料で落とせます。RTMP や F4V/FLV の仕様書もあります。

RTMP は英語版の Wikipedia ページがやたらと詳しいので参考になります。

F4V/FLV Technology Center

[http://www.adobe.com/devnet/f4v.html]

RTMP の仕様書

[http://www.adobe.com/devnet/rtmp.html]

Wikipedia ページ

[https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol]

Adobe Media Server のドキュメントも役立ちます。ちなにみ Adobe Media Server は 5 接続までしかできない Starter バージョンが無償で落とせるので動作確認などに使えます。

最終的にはオープンソースソフトウェアのソースを見ないとわからないところもあります。

Adobe Media Server

[http://www.adobe.com/jp/products/adobe-media-server-family.html]

Red5 Media Server

[http://red5.org/]

RTMPDump

[https://rtmpdump.mplayerhq.hu/]

RTMFP の方は RFC になっています。

RTMFP の RFC

[http://www.rfc-editor.org/info/rfc7016]

5. Microsoft のプロトコル仕様書

Microsoft はプロトコル仕様書をまとめて公開しているので探しやすくて助かります。

ASF の仕様書は Microsoft のサイトから落とせるはずなのですが、普通に探すと Not Found に飛ばされて見つかりません。頑張って探すとたまに残ってるのが発見できるのでそこから落としましょう。

Windows Media HTTP Streaming

[https://msdn.microsoft.com/en-us/library/cc251059.aspx]

Smooth Streaming Protocol

[https://msdn.microsoft.com/en-us/library/ff469518.aspx]

Advanced Streaming Format

[https://msdn.microsoft.com/en-us/library/cc251289.aspx] の下の方の ASF のリンク

6. WebM のサイト

WebM はサイトに情報がまとまってていいですね。VP8 についての各種資料も置いてあります。

WebM のサイト

[http://www.webmproject.org/]

7. Matroska のサイト

コンテナの Matroska のサイトです。仕様が書いてあります。

Matroska のサイト

[https://matroska.org/]

8. Xiph.org のサイト

Vorbis だの Opus だの作ってる Xiph.org のサイトです。ソースやら資料がまとまっています。

Xiph.org のサイト

[https://xiph.org/]

9. HTTP Live Streaming

HTTP Live Streaming の仕様は RFC のドラフトになっています。また、Apple のドキュメントも役立つでしょう。

RFC ドラフト

[https://www.ietf.org/id/draft-pantos-http-live-streaming-20.txt]

Apple の HLS のサイト

[https://developer.apple.com/streaming/]

10. WebRTC / ORTC

WebRTC は WebRTC のサイトを見てもよくわかりません。中身については W3C の仕様書を見た方がいいでしょう。

ORTC についてはサイトを見ても仕様書を見てもよくわかりません。

WebRTC のサイト

[https://webrtc.org/]

W3C の WebRTC 仕様書

[https://www.w3.org/TR/webrtc/]

ORTC のサイト

[https://ortc.org/]

11. Alliance for Open Media

AOMedia についてはあんまり詳しい情報はありません。Wikipedia あたりを読んどくのがわかりやすいんじゃないでしょうか。

AOMedia のサイト

[http://aomedia.org/]

Wikipedia の AOMedia 1 のページ

[https://ja.wikipedia.org/wiki/AOMedia_Video_1]

12. インプレス標準教科書シリーズ H.264/AVC 教科書

インプレスから H.264/AVC 教科書という本が出ています。H.264 コーデックの内容だけでなく MPEG-2 TS や MP4 といった応用についても記載されていて、日本語で一通り把握できる便利な本です。

今は H.265 対応の H.265/HEVC 教科書というのも出てるのでそちらを参照するのも良いでしょう。

H.264/AVC 教科書の方は古いからか Web サイトが見つからなかったので ISBN だけ貼っておきます。

H.264/AVC 教科書 改訂 3 版 :

[urn:isbn:9784844326649]

H.265/HEVC 教科書

[http://book.impress.co.jp/books/1112101148]
[urn:isbn:9784844334682]

著者紹介

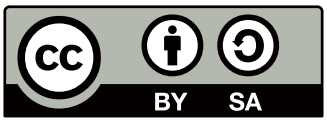
本文 : kumaryu

PeerCastStation とかいう P2P なライブ配信ソフトを作ってる人。ネットワークと動画はさっぱりわからん。

表紙 : ＃ペン

7144 お絵描き配信勢。

ライセンス



ライブ配信のなかみ

発行日： 2017 年 10 月 22 日 第二版発行
2017 年 4 月 9 日 初版発行
サークル名： あれくま
発行者： kumaryu
連絡先
Web: http://www.kumaryu.net/
メール： kumaryu@kumaryu.net

2017 あれくま