

Web-RTC による JetSon orin 低遅延 4K 映像配信

報告者：宮本 指導：小長谷

1. WebRTC のフレームワークについて調査

方法としては、2 つの候補があげられた。

①時雨堂「momo」

② AWS の KVS with WebRTC

そのため、以上の2点において、WebRTC の調査を行っていく。

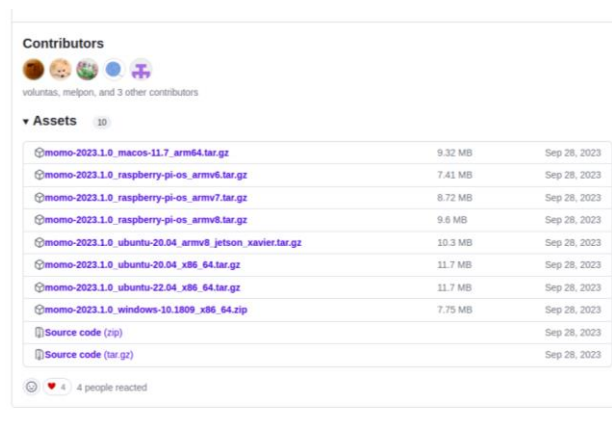
2. momo

(1)概要

momo は時雨堂が公開している OSS の WeRTC フレームワークである。

(2)使い方

1. 時雨道「momo」の github の[リリースノート](#)(スクロールして最新 Ver.の Assets 欄から探す)より、インストールを行う。対応した OS 等の.tar.gz ファイルを選択し、インストールする。



Contributors		
voluntas, melpon, and 3 other contributors		
▼ Assets 10		
📦momo-2023.1.0_macos-11.7_arm64.tar.gz	9.32 MB	Sep 28, 2023
📦momo-2023.1.0_raspberry-pi-os_armv6.tar.gz	7.41 MB	Sep 28, 2023
📦momo-2023.1.0_raspberry-pi-os_armv7.tar.gz	8.72 MB	Sep 28, 2023
📦momo-2023.1.0_raspberry-pi-os_armv8.tar.gz	9.6 MB	Sep 28, 2023
📦momo-2023.1.0_ubuntu-20.04_armv8_jetson_xavier.tar.gz	10.3 MB	Sep 28, 2023
📦momo-2023.1.0_ubuntu-20.04_x86_64.tar.gz	11.7 MB	Sep 28, 2023
📦momo-2023.1.0_ubuntu-22.04_x86_64.tar.gz	11.7 MB	Sep 28, 2023
📦momo-2023.1.0_windows-10.1809_x86_64.zip	7.75 MB	Sep 28, 2023
📄Source code (zip)		Sep 28, 2023
📄Source code (tar.gz)		Sep 28, 2023
👍 4 4 people reacted		

2. [USE.md](#) を参考に進めていく。※選択ブラウザに寄っては Google STUN が使えない場合がある。なので、[こちら](#)をテストでは利用すると動く。

(3)テスト

AYAME 等の NAT 超えが必要なテストにも成功、遅延もほぼない。しかし、多対一などの通信には SORA SFU の有料コンテンツを使わないといけない。

3. AWS の KVS

3.1. 概要

Kinesis Video Streams では、WebRTC の開発を助ける SDK ライブラリが作られている。その中で、組み込みデバイス用（ラズパイ、Jetson 等）の C 言語の SDK の `amazon-kinesis-video-streams-webrtc-sdk-c` を利用する。

3.2. 使い方

①まずは、ターミナルを開き、先述した Github より、パッケージのクローンを行う

```
git clone --recursive https://github.com/aws-labs/amazon-kinesis-video-streams-webrtc-sdk-c.git
```

②必要なライブラリのインストール

```
sudo apt-get install cmake m4 pkg-config libssl-dev libcurl4-openssl-dev liblog4cplus-dev  
libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-plugins-base-apps  
gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly  
gstreamer1.0-tools
```

③build の作成と Cmake と make 行う。

```
mkdir -p amazon-kinesis-video-streams-webrtc-sdk-c/build; cd amazon-kinesis-video-streams-webrtc-sdk-c/build; cmake ..
```

//build ディレクトリから make を行う。

```
make
```

④[AWS コンソール](#)に移動、アカウント作成 or ログインする。

⑤IAM ユーザーの作成（サービスタブから IAM を選択、ユーザー作成を）を行う。この時にポリシーの許可で「AmazonKinesisVideoStreamsFullAccess」をアタッチする。

※アクセスキーとシークレットアクセスキーを作成し、しっかりと記録

⑥ターミナルに戻り、AWS アカウントの認証情報、リージョンをセットする。

```
export AWS_ACCESS_KEY_ID=<AWS account access key> //アクセスキー
```

```
export AWS_SECRET_ACCESS_KEY=<AWS account secret key> //シークレットアクセスキー
```

```
export AWS_DEFAULT_REGION=<AWS region> //リージョン（東京：ap-northeast-1）
```

※=の前後にスペースがあるとうまくセットできない。

⑦サンプルを動かす。3.3 で解説

3.3 サンプルコードによるテスト

3.3.1 デフォルト（テストソース）テスト

①（build ディレクトリで）以下のコマンドを動かす

`./samples/kvsWebrtcClientMaster <チャンネル名>`

②コンソールを開き、Kinesis Video Streams アプリを選択、シグナリングチャンネルから<チャンネル名>のチャンネルを探し、開く。

③そこで、メディア再生ビューワーを開くを押して、以下のような映像が表示されることを確認する。



3.3.2 カメラ映像の受信の確認(ソフトウェアエンコード)

①gst-devicemonitor-1.0 によって、使うカメラのデバイス名と width,height,framerate をコピーしておく。
これは、接続したカメラなどの対応フォーマットを表示することができる便利機能。

②samples/kvsWebrtcClientMasterGstSample.c のファイルを編集する。

<https://github.com/aws-labs/amazon-kinesis-video-streams-webrtc-sdk-c/blob/1eaede838bc42e5ecb4ac6e6ea060a2e3e4e0076/samples/kvsWebrtcClientMasterGstSample.c#L168C21-L172C57>

-->このリンクの部分

以下の変更を加える。

・autovideosrc----->v4l2src device=/dev/video0（使いたいカメラデバイス番号）

…ここでは、デバイスカメラの選択を行っている。autovideosrc は使えるカメラを適当に取ってくるものだがカメラの指定をしたい場合は v4l2src を使ってデバイス指定を行う。カメラによっては v4l2src が使えないため、gst-devicemonitor-1.0 でそれぞれのデバイス情報の 1 番下に gst-launch-1.0 のサンプルがあるため、そちらを参照するとよい。

```
v4l2.device.version = 331663 (0x00050181)
v4l2.device.capabilities = 2225078273 (0x84a00001)
v4l2.device.device_caps = 69206017 (0x04200001)
gst-launch-1.0 v4l2src device=/dev/video2 ! ...
```

・videoconvert ! video/x-raw,width=1280,height=720,framerate=25/1

----[video/x-raw 形式]---->

videoconvert ! video/x-raw,width=720,height=580,framerate=30/1（メモした値を入力）

----[image/jpeg 形式]---->

image/jpeg,width=1280,height=720,framerate=30/1 ! jpegdec ! videoconvert

…ここでは、カメラからどのフォーマットで映像を取ってくるか（カメラからのデコード）を指定する。そのため、先程 gst-devicemonitor-1.0 で対応するフォーマットを調べた。（image/jpeg では、jpegdec という mjpeg 対応のデコーダーを使う必要がある）

③cmake .. && make を行い、以下を実行。メディア再生ビューワーで確認

./samples/kvsWebRTCClientMasterGstSample <channelname> <mediatype> <srctype>

・引数解説

<channelname>…シグナリングチャンネルの名前

<mediatype>…通信メディアタイプ

video-only（ビデオ転送）、audio-video（ビデオ、音声）、audio-video-storage（WebRTC のストレージ機能 ON）

<type>…使うソース

testsrc（テスト用の映像音声の転送）、devicesrc（gstreamer によるデバイス通信）、rtspsrc（rtsp 通信）

今回の場合、デバイスのカメラの映像転送のみなので以下を選択する。

<mediatype> ...video-only

<srctype>...devicesrc

(例)./samples/kvsWebRTCClientMasterGstSample test0 video-only devicesrc

④3.3.1と同様にコンソールで映像が映るか確認する。

3.4 ハードウェアエンコードによる低遅延高画質映像転送 (Jetson Orin, Nano)

前提条件 : jet pack ver. 5.1 シリーズの Jetson Orin, Nano で動作確認 (ハードウェアエンコードの要件)

①先程のプログラムを変更する、nvidia の対応した gstreamer のハードウェアエンコーダ、デコーダが使えるのでそちらに変更していく。

```
pipeline = gst_parse_launch(  
    "v4l2src device=/dev/video0 io-mode=4 !  
    image/jpeg,width=1920,height=1080,framerate=30/1 ! nvv4l2decoder mjpeg=1 ! video/x-  
    raw(memory:NVMM),format=I420 ! nvvidconv ! video/x-raw(memory:NVMM),format=NV12 ! "  
    "nvv4l2h264enc num-B-Frames=0 preset-level=1 ! h264parse ! "  
    "video/x-h264,stream-format=byte-stream,alignment=au,profile=baseline ! appsink sync=TRUE "  
    "emit-signals=TRUE name=appsink-video",  
    &error);
```

プログラムの流れ :

ビデオの取り込み (v4l2src) → GPU 上でデコード (nvv4l2decoder) → ハードウェアエンコードできる形にフォーマット変換 (nvvidconv) → ハードウェアエンコード (nvv4l2h264enc) → AWS 上への転送 (video/x-h264)

各処理の注意

•nvv4l2decoder :

nvvideo の GPU 内でデコードを行う。jpeg をデコードする際は mjpeg=1 の引数をつける。

•nvvidconv :

ビデオフォーマットの交換やスケーリング処理を行う。今回は、以下のように変更

I420--->NV12

(後述の nvv4l2h264enc において対応する NV12 のフォーマットに変える必要がある)

•nvv4l2h264enc :

H.264 形式での GPU によるハードウェアエンコード (引数はデフォルトのため、いらない)

②3.3.1 と同様に make と実行を行い、コンソールに映像が映ることを確認する。(必要に応じて、nvv4l2h264enc の引数部分で bitrate=???で指定する)

ビットレートに関しては[こちら](#)のサイトを参照した。

3.5. 全画面による確認

①こちらから、Web ページ (Java 系) 用 SDK の amazon-kinesis-video-streams-webrtc-sdk-js を git clone する。

Git clone <https://github.com/aws-labs/amazon-kinesis-video-streams-webrtc-sdk-js>

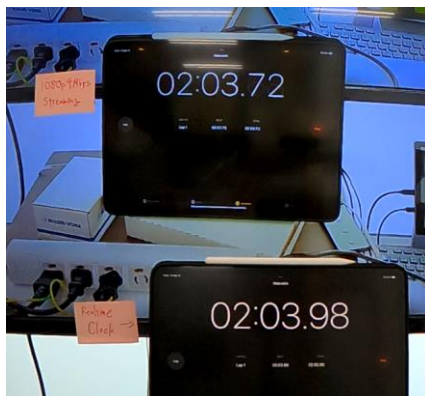
②この中の examples/Viewer.js、examples/Master.js、examples/index.html を編集し、それぞれで 1280→ (指定したい width) と 720→ (指定したい height) に変える。

③examples/index.html を開き、リージョン、アクセスキー、シークレットアクセスキー、チャンネル名を記入し、「start viewer」ボタンをクリック

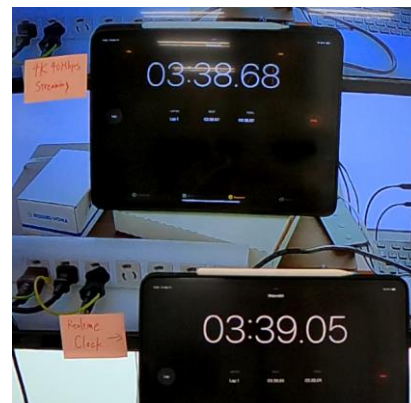
④右側の「From Master」からの映像が映るか確認する。

3.5 実施結果

社内 4 K ディスプレイにて、1080p(フル HD)と 4 K 画質での映像転送遅延テストを行った。以下が実験結果である。ビットレートはそれぞれ 4Mbps、40Mbps である。



1080p(フル HD) : 遅延 260ms



2160p(4K) : 遅延 370ms

遅延も 0.5 秒もなく、動作した。（しかし、4 K 画質が出ているかは USBWeb カメラではわからない）（データとしては 4 K 画質）手などで動きを加えても特に遅延が増大する様子はない。

また、Master 側のデバイスに影響を受けるらしく、Jetson から受け手：Windows ではうまく動作しないことがある。