ゲーム目線のインフラエンジニア
IETFにおけるQUIC, HTTP/3動向

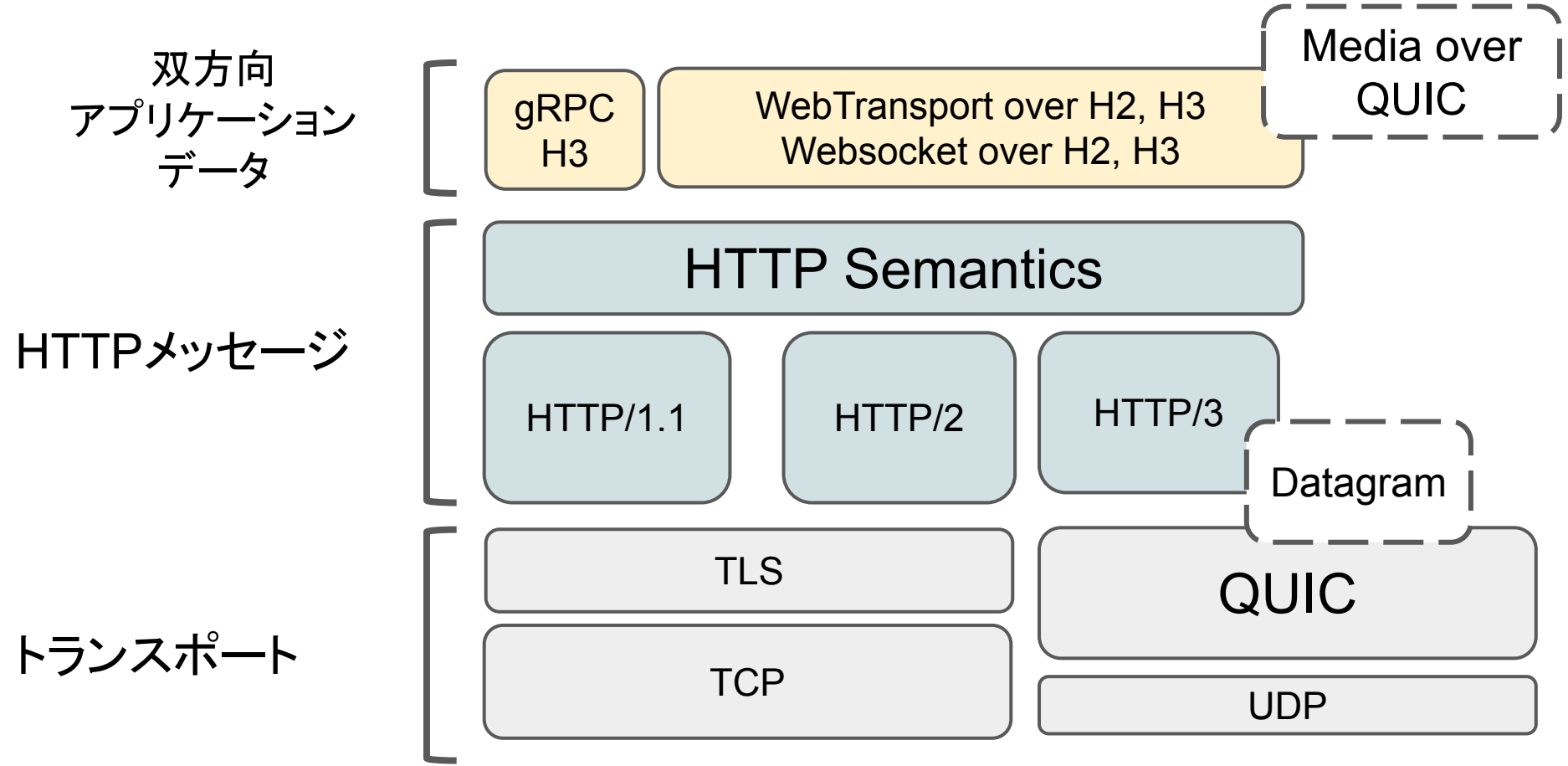
自己紹介

- ゆき (@flano_yuki)
- インフラエンジニア
- 興味/趣味
  - Web技術
  - HTTP, QUIC, TLS, WebTransport
  - 標準化 (IETF, W3C)
- ブログ
  - https://asnokaze.hatenablog.com/

今日のお話

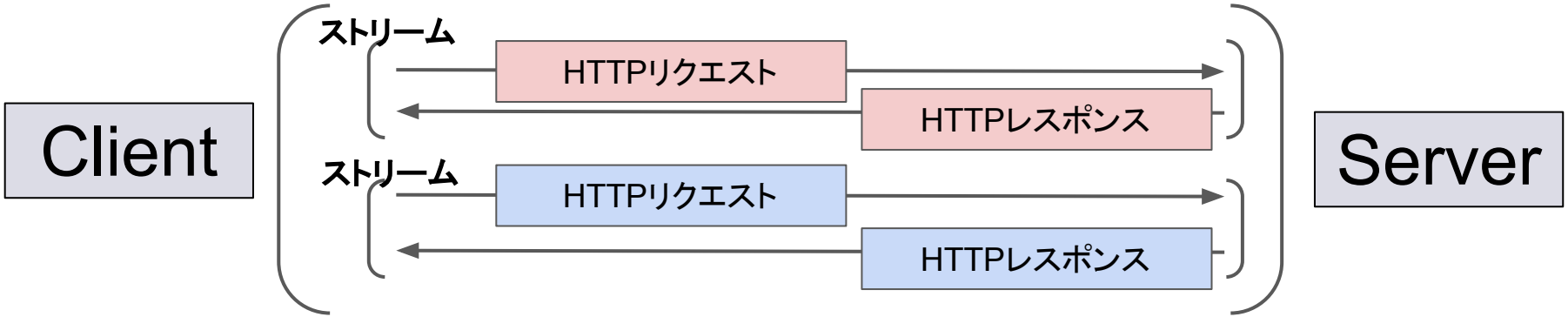
- 「プロトコル仕様好き勢」として、IETFでこんなこと話されているよというお話
- ゲーム目線を少しだけ入れてる

全体像 (アプリケーション)



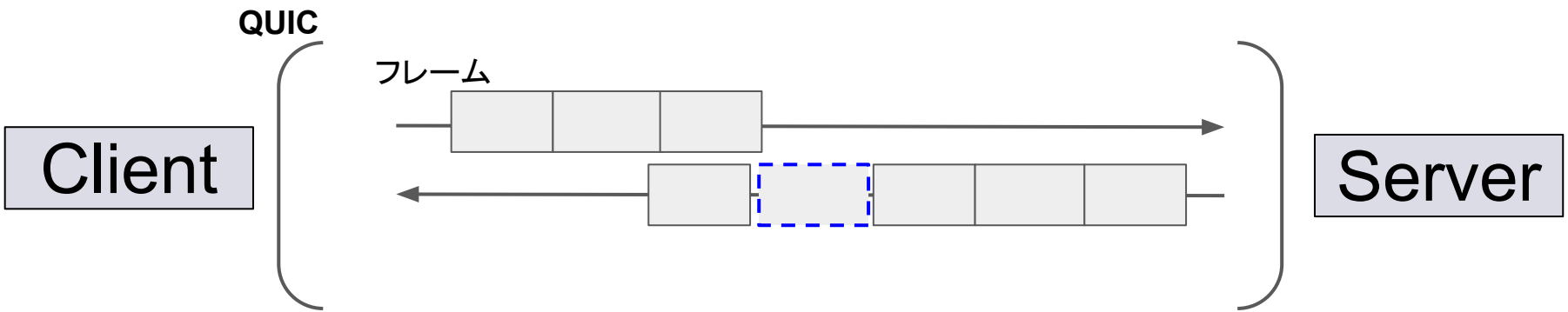
QUIC関連

- RFC 9000 QUIC
  - UDPベースの信頼性のある通信
    - アプリケーションデータはReliable + Odered
  - コネクションマイグレーション
  - HoLBの回避



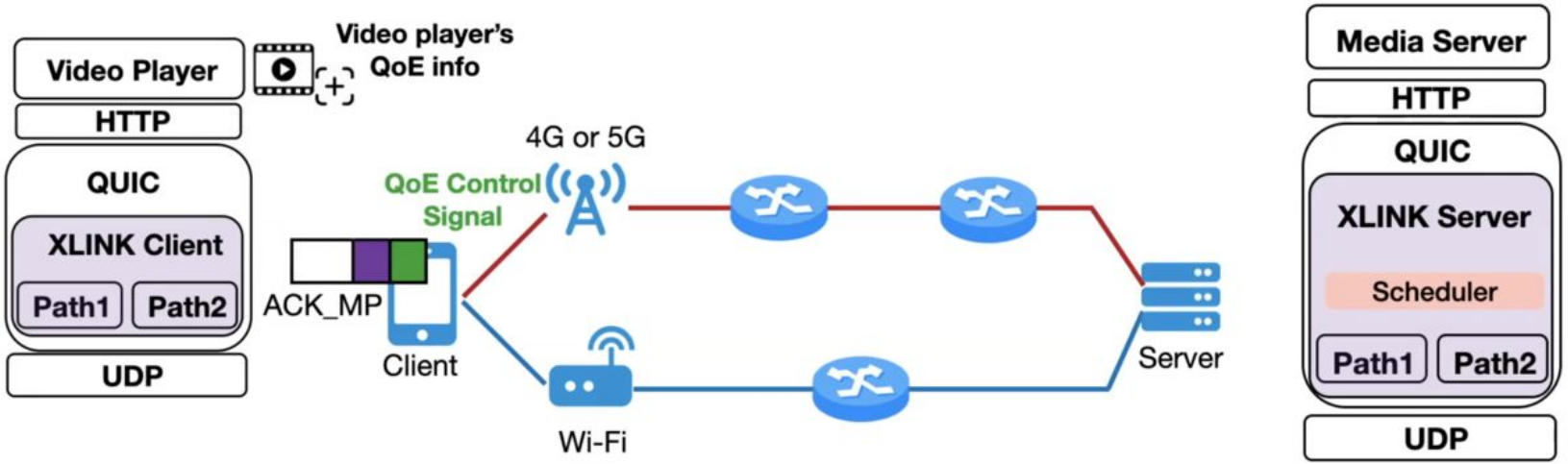
QUIC関連

- RFC 9297 Datagram
  - 信頼性のないアプリケーションデータ送信ができる
    - パケットの順番が入れ替わっても受け取った順に処理できる
    - パケロスしても送り直さなくていい



QUIC関連

- Multipath QUIC
  - Multipath TCPのように、複数の通信経路を使って通信する。輻輳やRTTを考慮したスケジューリング

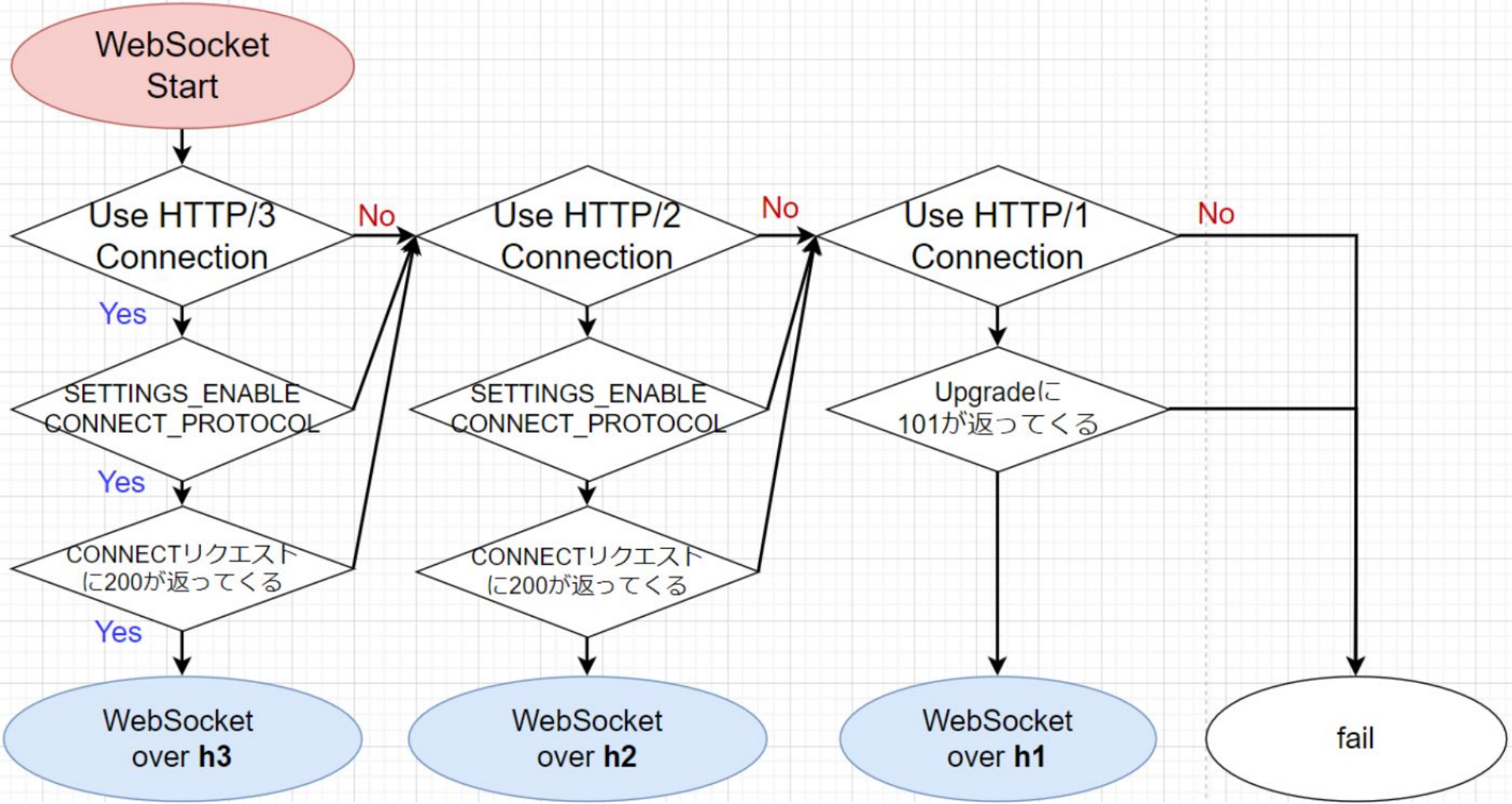


QUIC関連

- P2P QUIC
  - どちらがactive (client)とpassive (server) の役割を担うか
  - 証明書のフィンガープリント
- NATトラバーサルQUIC
  - Proxyを経由して一旦コネクションは確立したあとに、そのコネクション内でICE相当の処理をして
    コネクションマイグレーションで直経路に切り替える

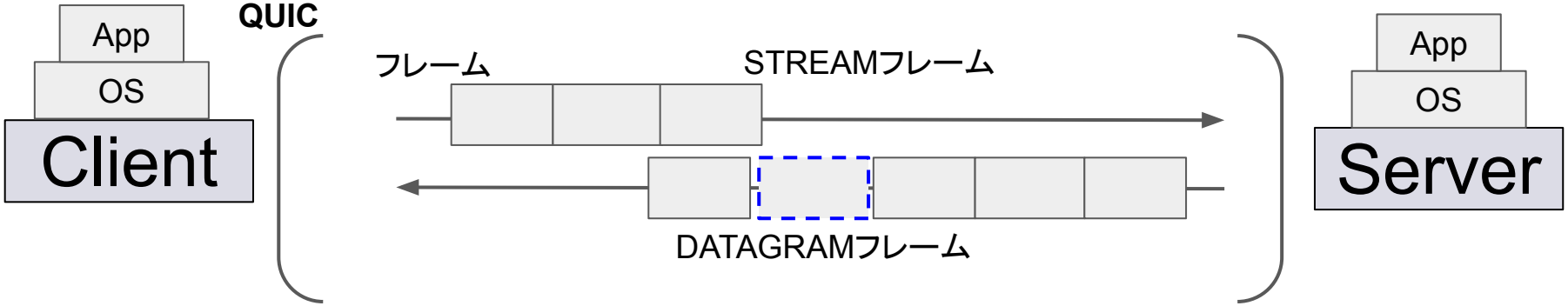
WebSocket over HTTP3

- WebSocketは仕様上、HTTP/3に対応している (RFC 9220)
  - SETTINGS_ENABLE_CONNECT_PROTOCOL が定義されているかだけの問題
- (後述) コネクション確立後に WebSocketが使えるかどうかリクエストを送るまで分からない
  - draft-momoka-httpbis-settings-enable-websockets の話


WebSocket Start
Use HTTP/3 Connection
No
Yes
SETTINGS_ENABLE CONNECT_PROTOCOL
Yes
CONNECTリクエストに200が返ってくる
Yes
WebSocket over h3
Use HTTP/2 Connection
No
SETTINGS_ENABLE CONNECT_PROTOCOL
CONNECTリクエストに200が返ってくる
WebSocket over h2
Use HTTP/1 Connection
No
Upgradeに101が返ってくる
WebSocket over h1
fail

WebTransport

- WebSocketの次世代版、HTTP/3で動作する
  - Chromeでは実装が進んでる
- 素のQUICでは、アプリケーションデータは必ず再送されます
  - パケットロスしても再送を必要としない通信を行いたい (例: ライブ動画)



Bidirectional Communication on the Web (proposed)

	Client-Server	Peer-to-peer
Reliable and ordered	WebSocket (also WebTransport!)	RTCDataChannel (WebRTC)
Reliable but unordered	WebTransport	
Unreliable and unordered		

https://datatracker.ietf.org/meeting/106/materials/slides-106-webtrans-webtrans-bof-slides-03

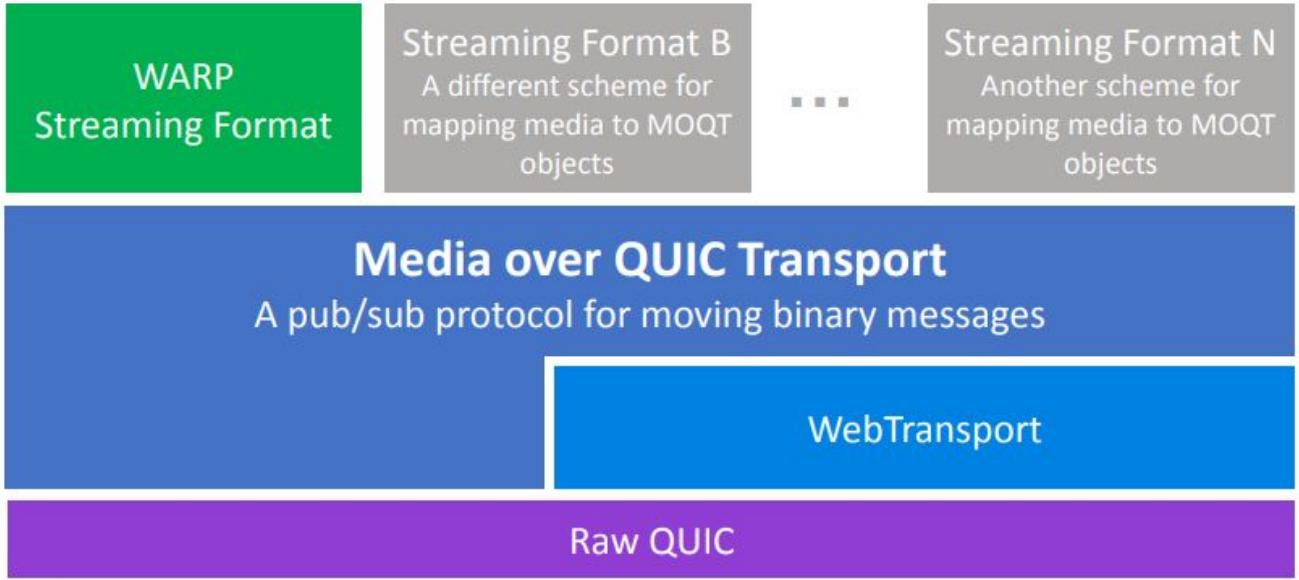
WebTransport

- QUICの ”DATAGRAM拡張” という機能を利用し、パケロスしても再送を必要としないアプリケーションデータをやりとりできるようになる
- Twitchとかはこれを、配信者からのライブ動画のアップロードに利用している
- (先述の通りフォールバック先として、WebTransport over h2もある)

```
session.SendMessage([]byte("hogehoge"));
```

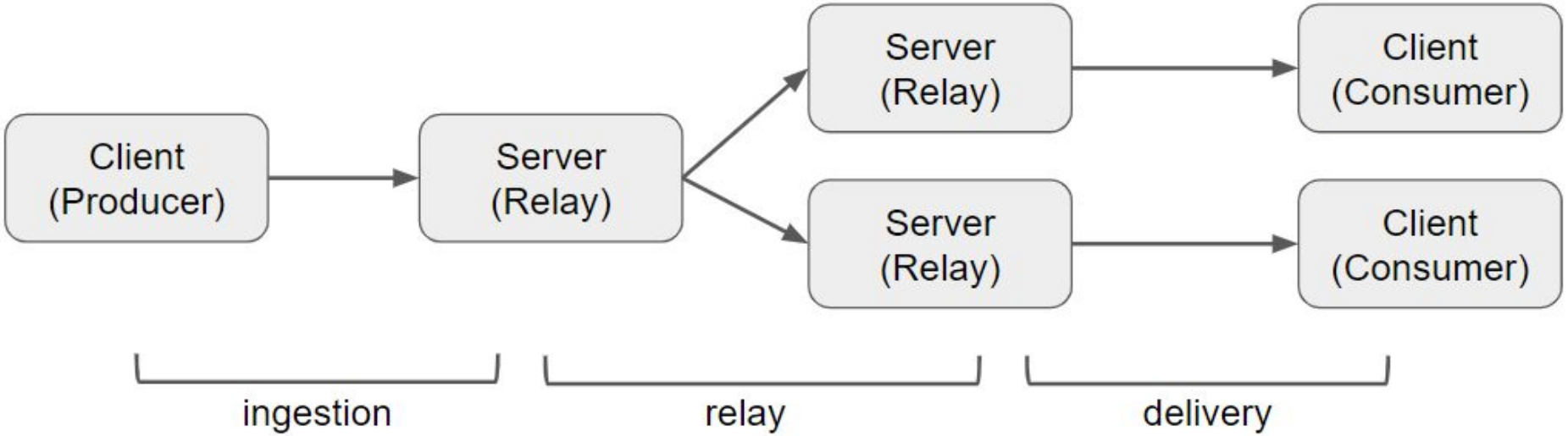
Media over QUIC

- Twitch, Meta, Google, Cisco, Akamaiなどの方々を含め標準化中
- QUIC上でメディアデータを流す



Media over QUIC Transport

- 動画のアップロード (ingestion)
- 動画の視聴 (delivery)
- CDNなどによる中継 (relay)



ゲーム系インフラエンジニア目線としておもうこと

ゲームの固有

サーバクライアントの環境として

- クライアント + サーバの実装が自分たちであるため、クライアントからサーバのcapabiityを事前知識として与える事ができる
  - WebSocket over HTTP/3のネゴシエーションの問題はない
  - ただしネットワーク環境の問題があるので、フォールバックの話はある
- Webブラウザの事を考えなくて良い
  - WebSocket, WebTransportを無理して使わなくても、、、QUIC直でいいのかもしれない
- サーバ環境としてクラウド環境であるため、**マネージドLBのサポートがないと辛い**

ゲームの固有

- クライアントのフォールバックとかは自分で考えなきゃいけない
- リアルタイムコミュニケーションソリューション的には、SaaS/SDKで隠蔽されるなら、ゲームを作る上では結構なんでもいい
  - ゲーム開発者からすると、独自プロトコルでも気にならない
  -

	やり取り	ブラウザ	クラウドLB	Head of Line	Datagram	マイグレーション	フォールバック先	平文
HTTP/2	req/res	◎	◎	✖	✖	✖	不要	可
HTTP/3	req/res	◎	◎	◎*(多重化)	✖	◎	必須	不可
Websocket h2	双方向	◎	？	✖	✖	✖	不要	可
Websocket h3	双方向	？	？	◎*(多重化)	✖	◎	必須	不可
Webtrans h3	双方向	◎	？	◎*(多重化)	◎	◎	必須	不可
QUIC	双方向	✖	？	◎*(多重化)	◎	◎	必須	不可

- 通常のWeb APIアクセスは HTTP/3 使って良い
- 双方向部分では選択肢は悩ましい
- フォールバック先を考える

HappyEye Ball v3

- HappyEye Ball とは、IPv4 / IPv6を試行してコネクションが確立したものを使う仕組み
- v3では(IPv4, IPv6) × (TCP, QUIC)を試すようになっている
  - 通常ユースでは HTTPSレコードも引くけど、アプリは普通に試行すればいいよね