ゲームリアルタイム通信プロトコル
第三回会合

議論の整理

リアルタイム通信まわりの現状整理

- プロトコルの整理
- 実装の整理
  - ライブラリ(OSS)
  - ゲームエンジン
  - 通信エンジン
  - プラットフォーム

現状整理：プロトコル

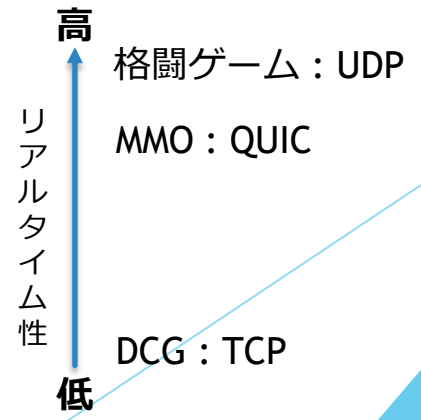
プロトコルに関するまとめ

後述する「必要な要素」と「主要なプロトコル」について今回協議した結果をまとめて以下を作成予定

- 「必要な要素」を各プロトコルがどのくらい満たしているかを示す早見表

	疎通性	暗号化	再送制御
TCP	○	△	○
UDP	△	△	×
QUIC	△	○	○

- ゲームジャンル毎に「重要な要素」や「現状向いているプロトコル」についてまとめる
  - こちらも一覧表のようなものを作成予定



プロトコル：ゲームのリアルタイム通信に必要な要素の整理

- 疎通性
- 暗号化
- 難読化
- 制御アルゴリズム
- 到達保証・リオーダー
- 再送制御
- 優先度
- Connection Migration
- Multipath
- 負荷

プロトコル：疎通性

- ミドルボックスにドロップされない等プロトコルそのものの疎通性
  - P2P における接続性とは別
- WAF やロードバランサ等に広く対応している
  - 別の指標として扱った方が良いかも？
- フォールバックの話もある
  - QUIC on Stream

プロトコル：暗号化

- 第二回記録より
  - 通信データを暗号化する
  - 第三者からの盗聴や成りすまし、改ざんを防止する
- 必要な機能
  - 何らかの単位で暗号化の有効無効を切り替え可能
  - 暗復号が高速に可能
  - パケットロスやリオーダーに強い
  - 安全性が定量的に評価されている
- もう少し大枠のセキュリティとした方が良いかも
  - プロトコルそのものの脆弱性への対策等

プロトコル：難読化

- 第二回記録より
  - ドライブやメモリ等のローカルに保存されるゲームのデータそのものを暗号化する
  - ユーザ(通信者)本人からのデータの不正利用やチートを防止する
- プロトコルレイヤーの仕様ではないので対象外とした方が良さそう

プロトコル：制御アルゴリズム

- パケロス率や RTT を加味する BBR のような(輻輳)制御
  - ほとんどのゲームはレイテンシ重視のはずなので、従来の Delay-based 輻輳制御を発展させたようなアルゴリズムが求められる
- TCP　vs UDP の流れで送信単位がストリーム単位化パケット（メッセージ）単位かという観点で五ラれがちだが、現代のインターネット的にはもう一歩踏み込んだ形で議論すべき
  - 例：遅延感知型(delay-sensing CC) 輻輳制御
    - 第一回議事録参照

プロトコル：到達保証・リオーダー

- 何らかの単位で到達保証・リオーダーの有効無効を切り替え可能
- 到達保証が無い場合も ack が返ってくることを選択可能

プロトコル：再送制御

- より素早く効率的にパケットロス時に再送が可能
  - パケットロス検出と一体
- パケットロス率や RTT の計測
  - 個別の項目にすべきかも

プロトコル：優先度

- ストリーム・メッセージ単位で優先度を変更できる
- 全ての通信を一つのプロトコルで賄えるとプロトコル側に優先制御処理を任せられて楽

プロトコル：Connection Migration

- モバイル端末でのゲームプレイには必要

プロトコル：Multipath

- 実装(OS 含む)・デバイス共に準備ができていないが、ある分にはプラス要素
- 必要(必須)かと言われると微妙……

プロトコル：負荷

- OSの最適化等も含んだ負荷、軽い方が当然良い
- プロトコル仕様かと言うと違うが選定基準としては重要な要素なので入れてある
  - 別の指標として扱った方が良いかも？
    - (別の話になるが)コンシューマへの移植容易性とかそういう指標もあって良いはず

ゲームで採用されている主要なプロトコル

- TCP
- UDP
- RUDP
- SCTP
- HTTP (1 - 2)
- WebSocket
- gRPC
- QUIC
- HTTP/3
- WebTransport

プロトコル：TCP

- ロストしてはならないデータ(課金やアイテム)の送受信に使われるケースが多い
- リアルタイム性が低いゲームの場合、TCPやHTTPで十分なケースも数多くある
- 疎通性が高いので(HTTPと共に)最終手段としてフォールバックされる利用されることも多い

プロトコル：UDP

- 座標データのような投げ捨てて良いデータの送受信に使われるケースが多い
- 疎通性は低くはない(むしろマイナープロトコルに比べれば高め)がそれでも疎通しないケースがあるのでTCPと併用されるケースもしばしば見受けられる

プロトコル：RUDP

- TCPの各種機能は要らないけど信頼性が欲しい場合の送受信に使われる
- 信頼性部以外にも暗号化対応等ゲーム向けに色々こねくり回して、各社秘伝のオレオレRUDPを完成させているという噂も絶えない
- 現在海外も含めて多くのリアルタイム通信ゲームがRUDPを採用している

プロトコル：SCTP

- CEDEC 2023で採用についての言及あり
  - https://github.com/TakeharaR/game-realtime-protocol/blob/main/cedec2023/minutes.md
- SCTP over UDP で実装
  - 送信側でByteストリームを使いたくなかった
  - パケットレベルでサイズのコントロールをしたかった
  - SCTPは元々はストリームしかないのでそこを拡張している
  - UDP上にSCTPを構築したのは疎通性の高いプロトコルに乗りたかったから

プロトコル：HTTP/1.1, HTTP/2

- 採用についてはTCPとほぼ同じである
- リアルタイムゲームのデータに用いるにはオーバースペックな面もあるが、WAF等サーバサイドはHTTPの方が揃っているので採用はプロジェクト事情による
  - 実装コストや疎通面でもHTTPの方が優位である
- HTTPを採用するならWebSocket/gRPCの方が親和性が高いので、実際に生でHTTPが採用されているケースは少ない

プロトコル：WebSocket

- 採用についてはTCP, HTTPと同様である
- ゲームのリアルタイム通信はほぼ100%双方向通信なのでHTTPを採用するならWebSocketを採用することになる
- HTTP同様にサーバ側の取り回しが良いので、RUDP, gRPC に続いて採用が多い印象である

プロトコル：gRPC

- そこまでリアルタイム性が高くない且つ Unity 製のゲームが多いモバイルで採用が多い
  - MagicOnionの存在が大きい認識
- WebSocketとどちらを採用するかはプロジェクトの構成やターゲットプラットフォーム次第

プロトコル：QUIC

- 仕様策定のタイミングを考慮するとそろそろ採用事例が表に出てきそうな頃合いだがまだない
- プラットフォームとしてはGDK(Windows, Xbox), Android, iPhone, Macあたりは実装は存在している
  - マルチプラットフォームで簡単に導入可能なものがないのでゲーム業界的に採用が難しいというのはある
- QUIC(UDP)が疎通しない環境でのTCPへのフォールバックが実装が煩雑になりがち問題はQUICを採用しても解消されない
  - QUIC on Stream に期待

プロトコル：HTTP/3

- WebSocketやgRPCのバックボーンとしての活用が期待されているがOSSまわりでの実装進捗が芳しくない認識
- HTTP/2へのフォールバックが容易なマルチプラットフォーム対応の実装が登場すると普及は爆発的に広がりそうな印象はある

プロトコル：WebTransport

- IETFにて仕様策定中なので実採用例はほぼない
- 以下第一回で議題に上った話 (参考)
  - https://github.com/TakeharaR/game-realtime-protocol/blob/main/mtg001/minutes.md
- オーバースペック？
  - ゲーム通信にQUICを組み込んでおり、検証を進めていると 要らないな、と思い始めた
    - ゲームでは Reliable/Unreliable なストリームがあればその上の層は要らない
  - WebTransport レイヤーの若干のオーバーヘッドも気になる
    - WebTransport(quic) は Unreliable(DATAGRAM) な通信も一つの実装で完備できるのが良い
- WebTransport を止めて quic を採用した際の問題
  - quic が疎通しない環境での TCP へのフォールバックが実装が煩雑になりがち
  - 更に完全な P2P を諦めてリレーを挟んでいる場合、 TCP/UDP のリレーどうする問題もある
- WebTransport のメリット
  - ロードバランサの問題が少な目
  - URI でエンドポイント識別指定可能なので設定が楽
  - 動画とコントロールを同じコネクションに載せられるので同じ輻輳制御や優先度制御ができる

現状整理：実装

※補足：UE の MsQuic 対応だけ先行で共有※

ゲームエンジン

- Unity
  - (g)RPC, QUIC, HTTP/3のような最新のプロトコルに標準で対応していない
  - Unity Transport Packageというレイヤーはある
    - UDPベースでRUDP相当の機能
- Unreal Engine
  - RPCはあるがRUDPはない
    - EOS側は掘っていないのでそちらにあるかもしれない
  - QUICに対応中
  - HTTP/3に非対応

ゲームエンジン：Unreal Engine QUIC 補足

- MsQuic を使用した実装が用意されている
  - Engine¥Source¥ThirdParty¥MsQuic に配置されている
  - いつのバージョンからかは不明
- MsQuic.Build.cs を見る限り Windows/Linux/Mac までの対応
- ヘッダを見た感じとりあえず突っ込んでラッパした程度に見える

スケジュール(再掲)

2023年

当会号の発足 (CEDEC 2023)

2024年　**①ゲーム業界のリアルタイム通信プロトコルにおける課題を洗い出す ← イマココ**

2025年　②QUIC等標準のプロトコルを用いた場合の検討・議論・事例収集 ← ここが少しずれそう……

2026年

③ゲームのリアルタイム通信プロトコルの標準化を進める

2027年

2028年

※実際には①-③の要素はある程度平行で進む想定