SecHack365

遠隔操作環境構築システム

impromptu

アンプロンプチュ

開発駆動コース　仲山ゼミ　中川皓太

簡単に複数のリモートデスクトップを配置、管理したい

現在、手軽に使えるリモートデスクトップはあるものの、デスクトップを複数管理、共有するといった遠隔操作のシステムをゼロから構築するのは大変。
ネットワークやサーバ構築などの専門的な知識がなくても簡単に構築したい。
そこで、サーバ要らずで簡単に起動できる遠隔操作システムを構築できる「impromptu」(アンプロンプチュ)を開発。（impromptuはフランス語で即席、準備無しでという意味です。）

impromptuの構成

遠隔操作の通信技術にはWebRTCを使用

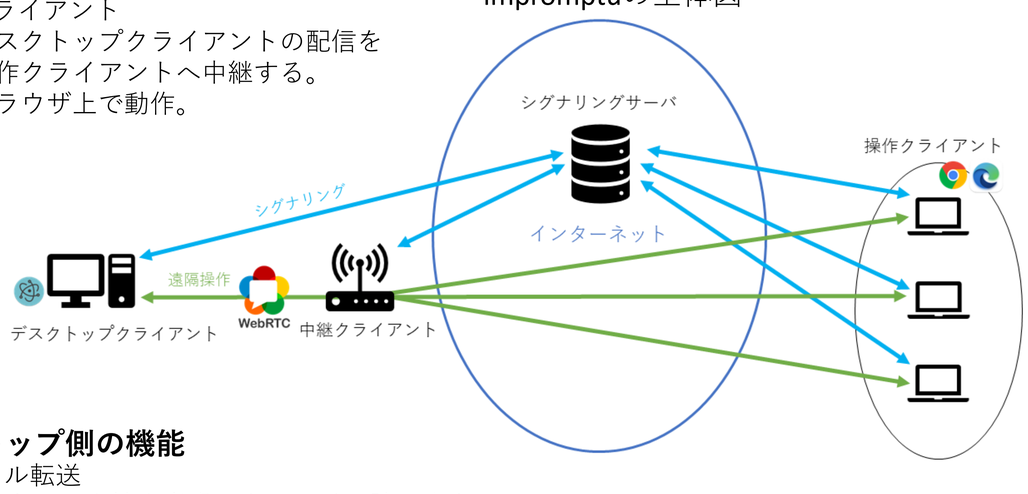
WebRTCではクライアント間で映像をリアルタイムに送受信できる通信技術であり、ブラウザ上でも使用することができる。
余計な操作クライアントツールをインストールせず、ブラウザさえあればどこでもデスクトップを操作できる。共有するデスクトップもサーバとして設定する必要はなし。

全体構成

WebRTCのシグナリングサーバを予め用意し公開することで、ユーザはデスクトップサーバ、ゲートウェイサーバを構築せず通信することができる。
３つのクライアントツールで構成

- デスクトップクライアント
  - デスクトップを共有する。（WindowsとLinux対応）
- 操作クライアント
  - 共有したデスクトップを操作できる。
  - ブラウザ上で動作。
- 中継クライアント
  - デスクトップクライアントの配信を操作クライアントへ中継する。
  - ブラウザ上で動作。

impromptuの全体図

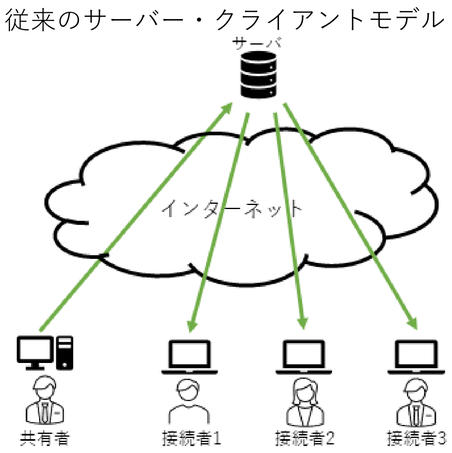


デスクトップ側の機能

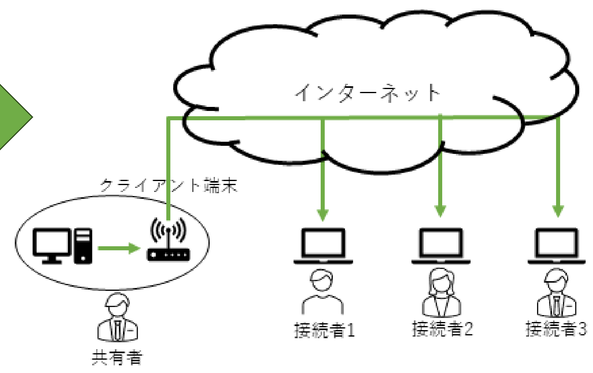
- ファイル転送
  - アトミック性を考慮したファイル読み書き。
- ウィンドウ単位での画面操作
  - 共有者が他の人に見られたくないウィンドウがあるといったプライバシー性を確保。
- コマンドラインで起動可能
  - SSHから実行やシェルスクリプトに組み込んで実行したりできる。
- 仮想ディスプレイ起動（Linuxのみ）
  - ディスプレイのない環境（例えばコンテナとか）でもGUIアプリを立ち上げられる。
  - 共有者が使用しているデスクトップ環境と隔離するので、マウス、キーボード操作が被るといったトラブルも防止。

１対Nの多数接続をクライアント端末だけで行う

従来のサーバー・クライアントモデル



impromptuはクライアント端末だけで行う



Impromptuで扱う遠隔操作の通信には、クライアント端末によるP2P方式を採用

しかし、リアルタイムの映像配信には一般的にサーバー・クライアントモデル（WebRTC SFU）が採用されている。

従来のサーバー・クライアントモデルの場合、サーバー上で映像・音声データを復号するため盗聴のリスクがあるが、E2EE(End-to-End Encryption)で対策はできる。また、各クライアントごとに最適な画質の映像を送信できる。

じゃあなんで、わざわざクライアント端末だけで行うの？

- **有事の際でも、事業者に通信内容を盗聴される可能性が低い**
  現在、世の中で使われている複数人のリアルタイムコミュニケーションツールのほとんどは海外製品であり、国産製品でシェアをとれているものはない。普段は安心して海外製品を利用できるが、仮に戦争等で日本が他国と対立した場合、対立国の事業者が好き勝手に日本ユーザーのプライバシーを無視して盗聴を行える可能性がある。
  E2EEがあるとはいえ、事業者が勝手にE2EEを無効にしたり、盗聴できるように脆弱性を仕込むことはできる。対策として自前でツールを用意することも挙げられるが、恐らく一部の技術力のある企業しかできない。
- **災害時における通信インフラの負荷軽減**
  震災などの大規模災害が発生した場合、通信インフラに甚大な被害を及ぼす可能性が極めて高い。被害が出ると生き残った通信インフラに通信負荷が集中し、インターネットへのアクセスがしずらくなる。サーバーを中継する方式だと必ずインターネットを介さなければならないが、クライアント端末間のP2P方式だと、一部LAN内で直接通信ができたりとインターネットへのアクセスをある程度抑えることができる。

クライアント端末間のP2P方式だと、<u>何かあったときでも</u>
安全に通信が行えるのではないかと私は考えている。

エンコーダの重複による負荷増大を対策

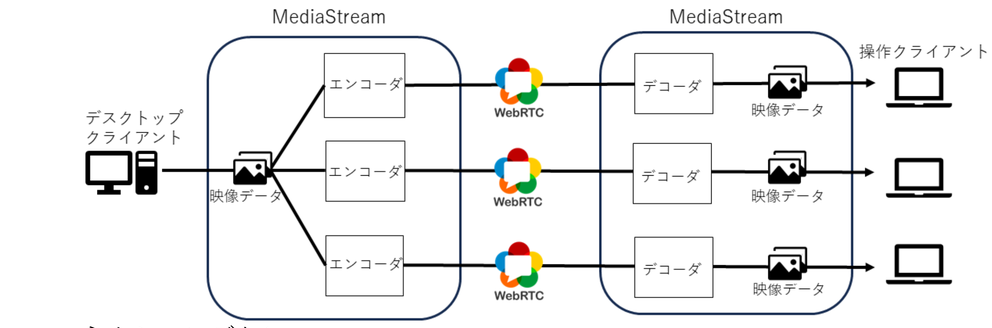
WebRTCの配信はクライアント端末には荷が重すぎる。

配信負荷を回避するためにWebRTC SFUがある。そのため、配信するのに十分な通信環境やマシンをユーザ側で用意しなければならない。

WebCodecsを使用し、エンコーダの負荷を軽減

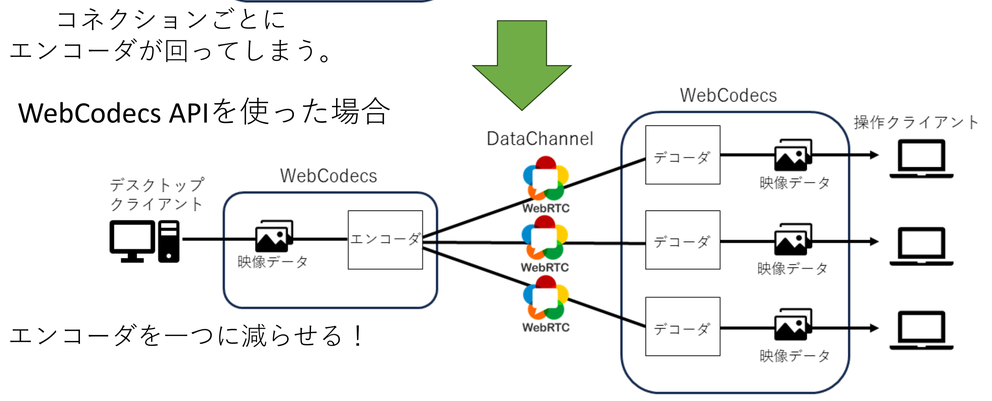
映像通信には従来、MediaStream APIを用いたSRTPベースの通信で行う。しかし、この方法だとコネクションごとにエンコーダが回るので、複数のコネクションを張ると負荷が大きくなる。そこで、WebCodecs APIを使用してエンコードを1回だけで済まし、負荷を抑えるように実装。エンコードしたバイナリデータは再送制御のないDataChannel上で送受信する。
中継クライアントもエンコードしたバイナリデータを横流しするだけのシンプルな実装にできる。

MediaStream APIを使った場合



コネクションごとにエンコーダが回ってしまう。

WebCodecs APIを使った場合

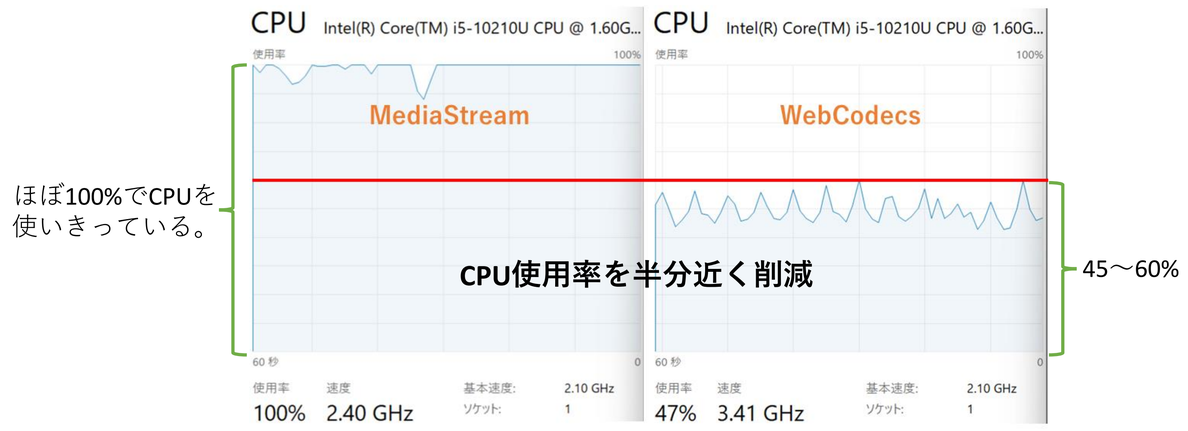


エンコーダを一つに減らせる！

MediaStreamとWebCodecsを用いた場合で中継配信に掛かるCPU使用率を比較測定

検証環境

- 動画（Big Buck Bunny：Full HD 1920x1080, 60fps）を5つの操作クライアントへ中継配信
- 中継クライアントのマシンスペック
  - CPU：Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz　2.10 GHz
  - メモリ：8.00 GB

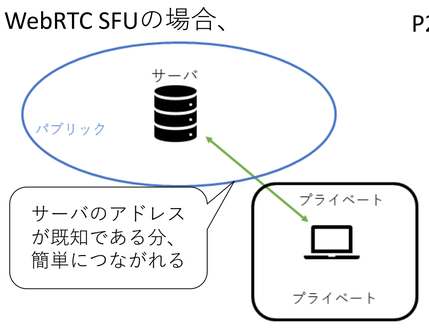


トラブルシューティングを容易に

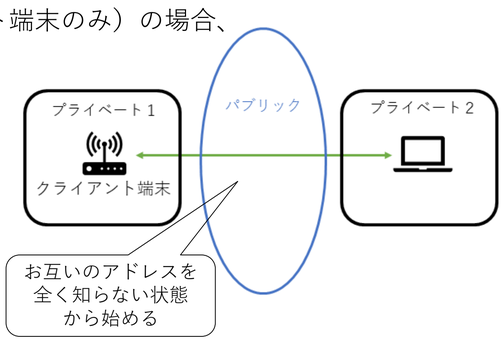
WebRTCの通信トラブルは、原因解明が難しい。

- クライアント同士だと接続確立が複雑。
- 通信品質はアプリだけでなく外部の問題(マシンスペックや通信環境等)からも影響を受けやすい。

WebRTC SFUの場合、



P2P（クライアント端末のみ）の場合、



トラフィック監視機能を実装し、接続状況を可視化

中継クライアントでWebRTCの接続状況を表示

- WebRTCの統計API(RTCIceCandidatePairStats)から情報を取得する。
- 映像の品質が悪い時、マシンスペックが足りないのか、通信経路が悪いのかを判断しやすく。
- リアルタイムで情報を更新。

実際に監視している様子

開発駆動コース仲山ゼミの方々をはじめ、本作品の開発に関わってくださった皆様に感謝を申し上げます。ありがとうございました。