Using QUIC to Traverse NATsの実装

Arch B1 Kota

概要

現在IETFで議論されているドラフト「Using QUIC to Traverse NATs[1]」にて提案されている、**QUIC上でNAT越えを行いP2P通信を確立する2つの手法を実装**し、QUICへの理解と実装力を深める

背景と目的

- 従来のP2P通信のほとんどはUDPで行われている
- 2021年にQUICプロトコルのバージョン1が公開され，普及してきている
- P2P通信を扱うアプリケーションは特に，コネクションマイグレーションなど接続の継続性を持つQUICの恩恵を受けやすい
- Using QUIC to traverse NATsでP2P通信 over QUICの手法が二つ提案された
- しかし現状それらの実装が存在しないため，本稿で実装を行う
- 将来的にQUIC周りのプロトコル開発に関わるための知識と実装力をつけるという目的も含む

先行事例

- Interactive Connectivity Establishment (ICE) [2]
  - UDP上のP2P通信において、最適なアドレスとポート、NAT越えの手順、失敗時のフォールバックなどを定義したプロトコル
- TCP Candidates with ICE [3]
  - TCP上のP2P通信におけるNAT越えの手法
- P2P QUIC [4]
  - QUIC上でICEプロトコルを作るためのドラフト
  - 今月ドラフトとしては期限切れになった

提案されている手法

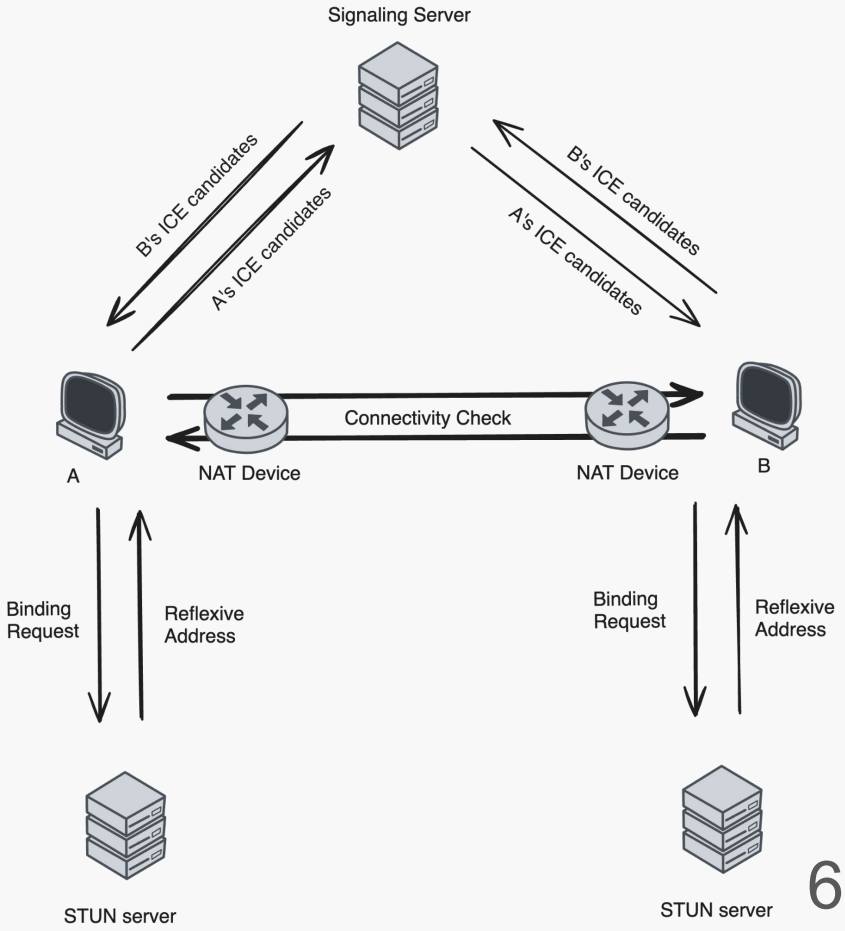
Using QUIC to traverse NATsで提案されている以下の二つの手法を説明する

1. **ICEプロトコルを用いてP2P通信を確立し、その上でQUICのセッションを確立する**
2. **QUICのみを使ってP2P通信を確立する**

手法1: ICEプロトコルを用いて接続を確立する

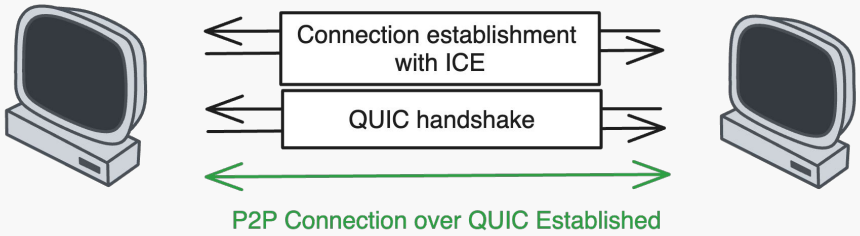
ICEプロトコルでまずUDP上でP2P通信を確立する

- NATデバイスの背後にあるA,B
- 仲介サーバーを通して候補となるアドレス(Candidates)を交換する
- 交換したCandidate全ての組み合わせに対して接続確認を行い、最適なペアを決定する
- 最適なペアが決定したらそのアドレスでP2P通信を確立する



手法1: ICEプロトコルを用いて接続を確立する

ICEプロトコルはUDPベースであり、同じくUDPベースであるQUICはICEで確立されたP2P通信をそのまま利用することができる



メリット

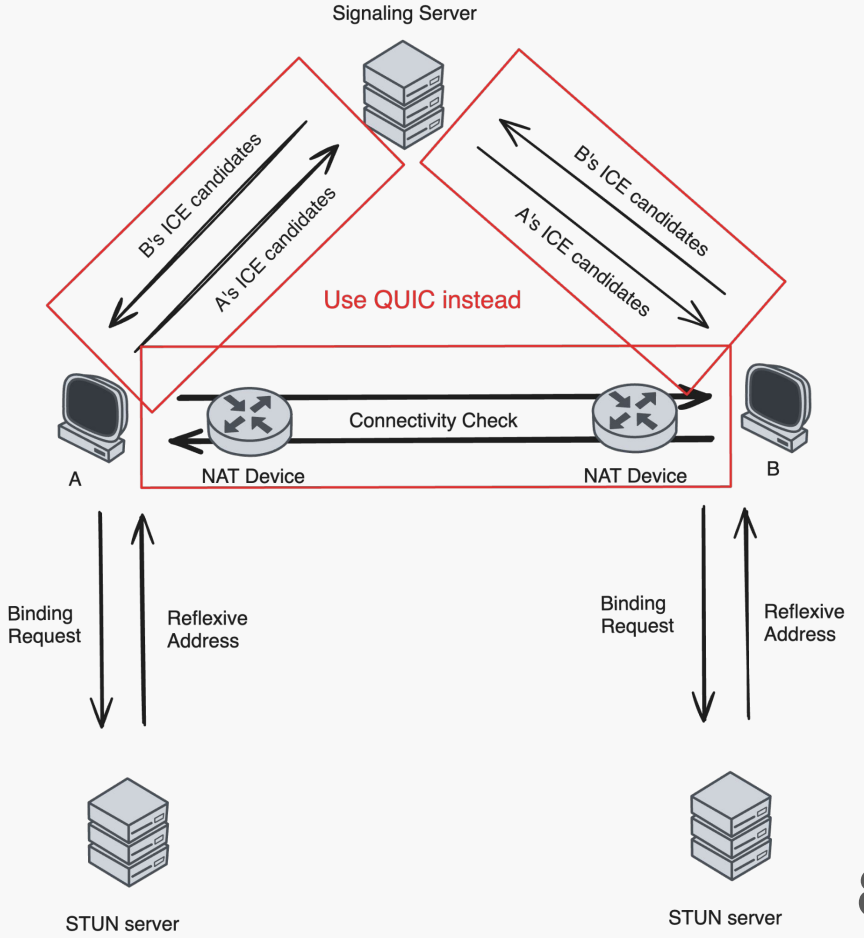
- 既存のICEプロトコルの実装が使える
- QUICの拡張ができない環境などで使える

デメリット

- 接続確立までに時間がかかる

手法2: QUICのみを使って接続を確立する

- 仲介サーバーとの通信もQUICを用いる
- 通信プロトコルが変わっただけで、最適なCandidateペアの選択アルゴリズムや交換の手順はICEを踏襲する
  - **ICE over QUIC**とも言える



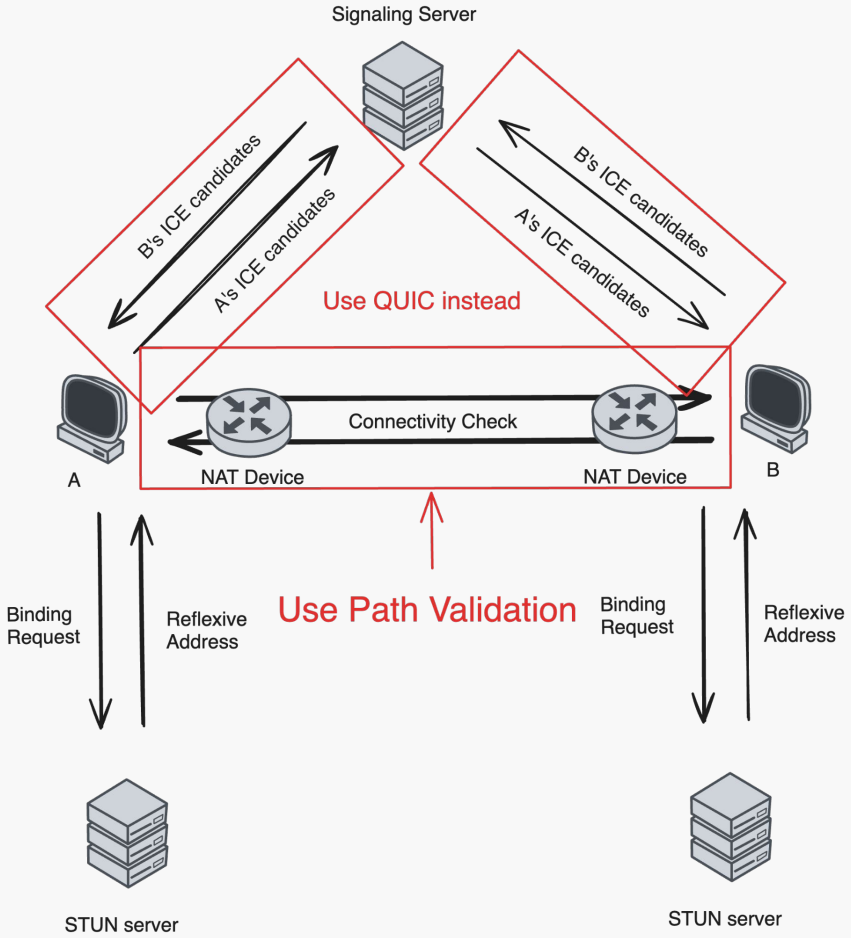
手法2: QUICのみを使って接続を確立する

- ペアの接続チェックに関しても、QUICハンドシェイクの一部である**Path Validation**を利用する

接続チェックとハンドシェイクを併せて行うことで手法1に比べて接続確立が早くなる

- Path Validationが完了したらそのままハンドシェイクの続きを行う

※本稿ではこの全ての実装が終わらなかったので手法2のうちの一部の実装に止まっている



実装: 手法1

使用した言語: Python
使用したライブラリ: aioice(ICEの実装), aioquic(QUICの実装)

ICEプロトコルで確立したP2P通信をQUICに再利用するためには**UDPソケットを再利用する必要がある**

- aioiceの実装を、接続が確立されたソケットを外部ライブラリに渡せるように変更した
- aioquicの実装も、外部からあらかじめ作ったソケットを引数として渡せるように変更した
- aioiceを用いてP2P通信を確立し、そのソケットを再利用してaioquicでQUIC通信を確立した

実装: 手法2

- aioquicの実装変更
  - 手法1同様、外部からソケットを引数で渡せるように変更する
  - サーバー側からもPath Validationを行えるように変更する
- ICEプロトコル全てを実装することができなかったため、**アドレス、ポート決め打ちでPath Validationを行う部分を今回の実装範囲とした**

実験

- 手法1, 2の実装を実際にローカルネットワーク内にある二つのピアで実行してみる
- 手法1に関しては通信確立までの速度の計測も行った
    - 最初のサーバーを送る前にタイムスタンプをスタートして計測した

実行環境

- ピア1: Mac OS 13.5 (Ventura)
- ピア2: Ubuntu OS 18.0.4 on Compute Engine
- 仲介サーバー: Heroku22 (Ubuntu 22.0.4)

ピアのNATタイプはどちらもAddress-restricted NAT (一度送信先になったIPアドレスのパケットは受け入れる)

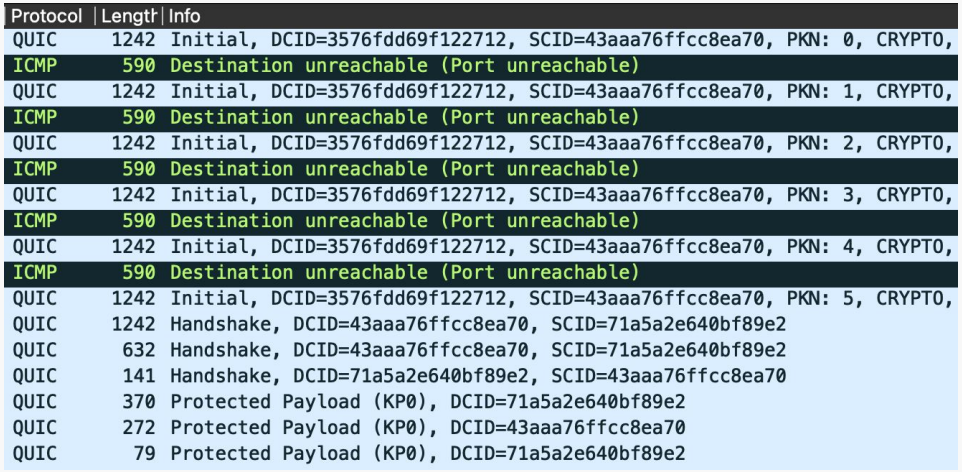
手法1の実験

ip.addr == 35.224.102.252

No.	Time	Source	Destination	Protocol	Length	Info
333	7.979682	192.168.11.22	35.224.102.252	STUN	130	Binding Request user: 1cOG:pdZU
372	8.134836	35.224.102.252	192.168.11.22	STUN	106	Binding Success Response XOR-MAPPED-ADDRESS: 106.72.33.225:17377
384	8.472820	35.224.102.252	192.168.11.22	STUN	134	Binding Request user: pdZU:1cOG
386	8.473954	192.168.11.22	35.224.102.252	STUN	106	Binding Success Response XOR-MAPPED-ADDRESS: 35.224.102.252:48794
391	8.628757	35.224.102.252	192.168.11.22	UDP	47	48794 → 61489 Len=5
396	8.629314	192.168.11.22	35.224.102.252	UDP	47	61489 → 48794 Len=5
397	8.651263	192.168.11.22	35.224.102.252	QUIC	1242	Initial, DCID=e489bc5113f7e54c, SCID=607413fe29c33fff, PKN: 0, CRYPTO, PADDING
412	8.842626	35.224.102.252	192.168.11.22	QUIC	1242	Handshake, DCID=607413fe29c33fff, SCID=9edac74e1b89670c
413	8.842627	35.224.102.252	192.168.11.22	QUIC	632	Handshake, DCID=607413fe29c33fff, SCID=9edac74e1b89670c
417	8.847908	192.168.11.22	35.224.102.252	QUIC	141	Handshake, DCID=9edac74e1b89670c, SCID=607413fe29c33fff
418	8.852628	192.168.11.22	35.224.102.252	QUIC	370	Protected Payload (KP0), DCID=9edac74e1b89670c
463	9.006987	35.224.102.252	192.168.11.22	QUIC	272	Protected Payload (KP0), DCID=607413fe29c33fff
465	9.008637	192.168.11.22	35.224.102.252	QUIC	79	Protected Payload (KP0), DCID=9edac74e1b89670c
466	9.009150	192.168.11.22	35.224.102.252	QUIC	75	Protected Payload (KP0), DCID=9edac74e1b89670c
480	9.164680	35.224.102.252	192.168.11.22	QUIC	72	Protected Payload (KP0), DCID=607413fe29c33fff

- STUNパケットを用いた接続性チェックの後，QUICでのハンドシェイクが始まり，その後通信が確立されている
- 30回実行したところ，通信確立にかかった秒数は平均約5.35秒，ICEは約4.80秒
  - **UDPでのP2P通信に比べて手法1は0.5秒ほど遅くなる**

手法2の実験



Protocol	Length	Info
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 0, CRYPTO,
ICMP	590	Destination unreachable (Port unreachable)
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 1, CRYPTO,
ICMP	590	Destination unreachable (Port unreachable)
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 2, CRYPTO,
ICMP	590	Destination unreachable (Port unreachable)
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 3, CRYPTO,
ICMP	590	Destination unreachable (Port unreachable)
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 4, CRYPTO,
ICMP	590	Destination unreachable (Port unreachable)
QUIC	1242	Initial, DCID=3576fdd69f122712, SCID=43aaa76ffcc8ea70, PKN: 5, CRYPTO,
QUIC	1242	Handshake, DCID=43aaa76ffcc8ea70, SCID=71a5a2e640bf89e2
QUIC	632	Handshake, DCID=43aaa76ffcc8ea70, SCID=71a5a2e640bf89e2
QUIC	141	Handshake, DCID=71a5a2e640bf89e2, SCID=43aaa76ffcc8ea70
QUIC	370	Protected Payload (KP0), DCID=71a5a2e640bf89e2
QUIC	272	Protected Payload (KP0), DCID=43aaa76ffcc8ea70
QUIC	79	Protected Payload (KP0), DCID=71a5a2e640bf89e2

- サーバー側の NATマッピングがなされていない最初の数回の接続試行はDestination unreachable で失敗している
- その後NAT 越えが成功し，通信が確立している

今後の計画

- 手法2: ICE over QUICの実装
  - 実際にQUICの上にICEプロトコルを実装し、接続確立まで行う
  - 手法1と速度、接続性、成功率について比較する
- 評価
  - 他プロトコル（TCP/UDPなど）でのNAT越えとも同様に比較する
- QUICの上に乗る他プロトコルの実装
  - 今回のaioquicの拡張実装を利用して他のQUICとP2Pに関するプロトコルやアプリケーションの実装を進める

結論

- QUIC上でP2P通信を確立する手法はこれまで仕様として提案されておらず、Using QUIC to Traverse NATsで初めて提案された
- 提案されている手法の実装はまだ公開されていなかったため、今回QUICへの理解の向上も兼ねて実装を行った
- 提案手法に完全に沿った実装は間に合わなかったが、QUIC上でのP2P通信は確認することができた
- 提案されている2つの手法の比較や定量的な評価は今後行っていく

参考文献

[1] Marten Seemann. Using QUIC to traverse NATs. October 2023.

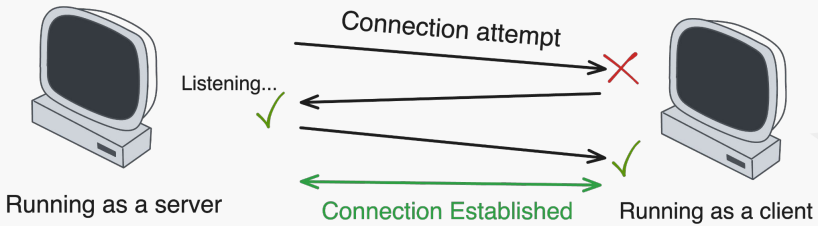
[2] A. Keranen, C. Holmberg, J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. RFC8445. July 2018.

[3] J. Rosenberg, A. Keranen, B. B. Lowekamp, A. B. Roach. TCP Candidates with Interactive Connectivity Establishment (ICE). RFC6544. March 2012.

[4] Peter Thatcher. P2P QUIC. July 2023.

appendix: 手法2の実装（WIP版）

> **4.1. Gathering Address Candidates**
> The gathering of address candidates is out of scope for this document.
> Endpoints MAY use the logic described in Sections 5.1.1 and 5.2 of [RFC8445],
> or use address candidates provided by the application.



サーバーがまず接続試行しNATマッピングしておく
その後クライアントから接続試行すると通信が可能になる