Using QUIC to Traverse NATs

IETF 118
Marten Seemann, Erik Kinnear

draft-seemann-quic-nat-traversal
QUIC v1 (RFC 9000)

- Assumes that the server is always publicly reachable
- Only the client might be behind a NAT
- Defines how to handle NAT rebindings
- Defines how a client can actively migrate to a different path
ICE (RFC 8445)

1. Peers gather candidates
2. Exchanges candidates between peers
   a. Match candidate pairs
3. Perform connectivity checks
4. Nominate candidate pair
5. Keeping paths alive

Figure 1: ICE Deployment Scenario
Purpose of this Draft

- Make it possible to use QUIC in a peer-to-peer setting
- Possible use cases:
  - Building block for WebRTC over QUIC
  - ... lots of other p2p protocols
But... do we need to do anything?

1. Use ICE to do all the NAT traversal
2. Run a QUIC handshake on ICE's nominated address candidate pair

- Requires running ICE
- Requires running a (non-QUIC) signaling server
- Lots of round trips
What if we do it in QUIC?

1. Use a proxied QUIC connection for signaling
   ○ for example: connect-udp-listen
2. Use QUIC path probing to create the NAT binding
   ○ Requires the server to send a probe packets
3. Then use QUIC connection migration
Step 1: Address Discovery

- The server sends all its addresses to the client
  - The draft defines an ADD_ADDRESS frame
  - This allows trickling of addresses
- No addresses are sent from the client to the server

Server Addresses:
- 192.168.13.37
- 10.0.0.42
- 17.42.10.89
Step 2: Address Matching

- Happens on the client side
- MAY use ICE's address matching logic

Client Addresses:
- 192.168.0.78
- 10.0.10.29
- 90.181.162.10
- 139.162.34.83

Server Addresses:
- 192.168.13.37
- 10.0.0.42
- 17.42.10.89
Step 3: Traversing the NAT

- Both peers send probe packets for each candidate pair
- If the hole punching is successful, a new QUIC path is established
- The client may now initiate QUIC Connection Migration

Candidate Pairs:
- 10.0.10.29 ↔ 10.0.0.42
- 90.181.162.10 ↔ 17.42.10.89
- 139.162.34.83 ↔ 17.42.10.89
Does this require QUIC Multipath?

It's not necessary. But potentially beneficial.

<table>
<thead>
<tr>
<th></th>
<th>QUIC v1</th>
<th>QUIC Multipath</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client can probe (multiple) paths</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Server can probe paths</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>
Open Questions

- Probing paths requires a lot of Connection IDs, which might clash with the `active_connection_id_limit`.
- Bandwidth requirement of path probing.
- Asking a peer to dial many addresses is an amplification vector.