Decentralized Hole Punching

Marten Seemann
Protocol Labs
marten@protocol.ai

Max Inden
Protocol Labs
max.inden@protocol.ai

Dimitris Vyzovitis
Protocol Labs
vyzo@protocol.ai

ABSTRACT

We present a decentralized hole punching mechanism built into the peer-to-peer networking library libp2p [1]. Hole punching is crucial for peer-to-peer networks, enabling each participant to directly communicate to any other participant, despite being separated by firewalls and NATs. The decentralized libp2p hole punching protocol leverages protocols similar to STUN (RFC 8489 [2]), TURN (RFC 8566 [3]) and ICE (RFC 8445 [4]), without the need for any centralized infrastructure. Specifically, it doesn't require any previous knowledge about network participants other than at least one (any arbitrary) node to bootstrap peer discovery. The key insight is that the protocols used for hole punching, namely address discovery and relaying protocols, can be built such that their resource requirements are negligible. This makes it feasible for any participant in the network to run these, thereby enabling the coordination of hole punch attempts, assuming that at least a small fraction of nodes is not located behind a firewall or a NAT.

1 INTRODUCTION

Consumer devices as well as computers in the corporate networks are often located behind a Network Address Translator (NAT) and / or a firewall. These devices usually allow (relatively) unobstructed access from within the network to the internet, but block incoming connections from computers on the internet to a local machine on the network.

While this network configuration provides security and privacy advantages, it poses significant challenges for connectivity in peer to peer (p2p) applications [6]. In most peer-to-peer networks only a small fraction of nodes are both well resourced and publicly reachable. The majority of nodes operates with limited resources and behind firewalls and/or NATs. Direct connectivity between these limited nodes is key for a peer to peer network to function well as a whole. Therefore, so called "hole punching" techniques have been developed to facilitate the establishment of direct connections between nodes located behind such NATs and firewalls. Recent measurements [?] found about 52 percent of nodes in the IPFS p2p network to be located behind a NAT.

Conventionally, hole punching needs central coordination servers running the STUN (Session Traversal Utilities for NAT) and coordination for the ICE (Interactive Connectivity Establishment) protocols. This creates reliance on centralized architecture, in two ways: first, somebody needs to run these coordination servers, and second, these servers need to be hard-coded / configured in the p2p application.

Decentralizing this infrastructure therefore makes the network more resilient against targeted attacks and against censorship attempts, while at the same time removing the necessity of maintaining these servers.

2 NATS AND FIREWALLS

NATs can be classified by the way they map addresses from nodes in the internal network to external (internet-facing) addresses. The terminology used in RFC 4787 [5] will be used in this paper. In general, a NAT is a device that maps an internal address tuple (IP and port) X:x to an external address tuple X':x'. The NAT type is determined by looking at the relationship between the external addresses X'_1:x'_1 and X'_2:x'_2 assigned by the NAT when connecting to external addresses Y_1:y_1 and to Y_2:y_2 afterwards (see figure 1).

- *Endpoint-Independent Mapping*: Packets sent from the same internal address X:x are mapped to the same address X':x', for all combinations of Y_1:y_1 and Y_2:y_2.
- *Endpoint-Dependent Mapping*: Packets sent from the same internal address X:x are mapped to different addresses X':x', either when sent to to a different address tuple, i.e. for all combinations of Y_1:y_1 unequal Y_2:y_2 (called *Address and-Port-Dependent Mapping*), or when sent to a different IP addresses, i.e. for Y_1 unequal Y_2 (called *Address-Dependent Mapping*).

NATs using *Endpoint-Independent Mapping* lend themselves to reliable hole punching. Nodes can rely on their external address X':x' to be stable, and advertise this address to other nodes. On the other hand, there is no reliable way to punch through NATs using *Endpoint-Dependent Mappings*; the difficulty lies in predicting the port the NAT will assign, which becomes exponentially harder with concurrent connection attempts.

For our purposes, in their most basic configuration firewalls are conceptually simpler than NATs. A firewall makes sure that packets originating from an address outside the network only pass through the firewall if a packet was sent to the same address from within the network before (within a certain time frame).

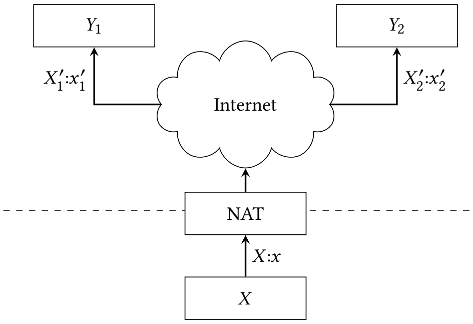

Figure 1: Address and Port Mapping (after [5]). We classify NATs by the IP addresses they assign for subsequent sessions originating from the internal node X to external nodes Y_1 and Y_2. For *Endpoint-Independent Mappings*, the external address X':x' will be the same, regardless of the external endpoint. For *Endpoint-Dependent Mappings*, the external addresses will differ.

3 DIRECT CONNECTION ESTABLISHMENT IN LIBP2P

libp2p uses stream-multiplexed connections, either by making use of QUIC streams, or by applying a stream multiplexer in the case of a TCP connection. Connections are always encrypted, and nodes verify each others' identities during a cryptographic handshake.

Application protocols running on top of libp2p request streams from the libp2p stack in order to exchange application data with a peer. At the same time, libp2p itself uses streams to run various libp2p-internal protocols.

In the following, we describe how we leverage various libp2p functionality in order to punch holes into NATs and firewalls.

3.1 *Identify* Protocol

Identify is usually the first protocol run on a newly established libp2p connection. It serves a variety of use cases in the libp2p stack, but for the purposes of this discussion, it provides functionality roughly similar to the STUN protocol (RFC 8489 [2]). The peer sends the observed address (along with other information) to its newly connected peer. This allows that node to discover its public IP address and port, and infer if it is located behind a NAT. By observing the reported addresses from multiple peers, the node can also infer it's located behind a NAT that uses endpoint-dependent or endpoint-independent mappings.

Using *Identify* instead of STUN provides multiple advantages: First, it is essentially free, as it reuses an existing connection. Second, it doesn't require any additional infrastructure or configuration, as the vast majority of the nodes on a libp2p network support this protocol. Third, the information can be assumed to be more reliable than what could be obtained from a STUN connection, as the connection leverages the same set of transport protocols as the hole punched connection.

3.2 *AutoNAT* Protocol

The *AutoNAT* protocol is a libp2p protocol introduced in 2018. It is used determine the reachability of a node. *Identify* cannot provide that information: Just learning that a NAT uses a certain IP address on outgoing packets doesn't necessarily imply that other nodes will be able to successfully dial that address from the internet.

Using *AutoNAT*, a node can request a peer to dial a new connection on a set of addresses. The peer then reports back if it succeeded in establishing a new connection, along with the specific address.

If these dial back attempts succeed on a regular basis, the node can conclude that it is not behind a NAT or firewall that blocks incoming connections. In the following, we call this a "public node". On the other hand, if these dial-back attempts regularly fail, the node can conclude that it cannot receive incoming connections (without prior coordination), and is called a "private node". Public nodes provide (limited) relay services to private nodes. Conversely, a private node might start searching and acquiring reservations with relay servers in order to coordinate hole punches to other nodes later on.

3.3 *Circuit v2* Protocol

Circuit v2 is a relaying protocol used primarily for the coordination of hole punching. The protocol was designed to be extremely lightweight, in the sense that running a relay server doesn't impose any non-negligible cost in terms of processing power or bandwidth. While this means that *Circuit v2* relays by default cannot be used as TURN-style (RFC 8656 [3]) relay servers, which relay the entire traffic between two nodes, it allows the vast majority of public libp2p nodes on the network to provide relaying services.

To achieve this minimal footprint, *Circuit v2* employs multiple strategies: First, private nodes wishing to use a relay's service need to obtain a reservation with that relay. Relay servers limit the number of concurrent reservations, and reject incoming reservation requests once the limit is reached. Second, every reservation is only valid for a certain duration and only allows the exchange of a bounded amount of data in a temporally limited relayed connection.

Once a reservation has been obtained, the private node makes sure to keep the connection to the relay server alive – after all, the relay would not be able to dial a connection to that node. In practice, a private node attempts to make reservations with a few relays at the same time.

After having obtained a reservation, the node advertises that it is reachable *via the relay server*. Other nodes are then able to first connect to the relay server, and then ask the relay to "relay" a connection to the private node. To do so, the relay server opens a new stream to the private node and notifies it about the incoming connection attempt. Once the private node agrees to establish a connection, the relay copies all data from the external node's stream onto this stream, and vice versa, effectively creating a bidirectional byte-stream between the two nodes.

The two nodes use this byte-stream to establish a virtual (encrypted) libp2p connection. This means that a malicious relay cannot intercept or alter data sent on the relayed connection. The worst it can do is to stop forwarding data, which is a scenario that libp2p nodes are well equipped to handle – connections on the internet regularly break for all kinds of reasons.

3.4 Hole Punch Coordination (*DCUtR* - Direct Connection Upgrade through Relay)

Nodes use the *DCUtR* protocol (Direct Connection Upgrade through Relay) to coordinate the establishment of a direct connection over a relayed connection. This protocol was designed to be extremely lightweight, under normal conditions the stream is only needed for 2 network round-trips and exchanges less than 500 bytes in each direction.

The protocol interaction is triggered once a private node accepts a relayed connection, that is considered the initiator of the protocol. At this point, it does not know if its peer is a public or a private node, but it does have the peer's addresses obtained through identify. In case the peer is a public node, no hole punching is needed: The initiator can simply dial a direct connection to the peer. This is called *Connection Reversal*, and depicted in figure 2.

If the direct connection attempt fails, the initiator concludes that its peer is most likely behind a NAT itself and and starts the hole punching procedure to establish a direct connection, as depicted in 3.

The first message exchanged is the *CONNECT* message. It contains a list of addresses that a node can use for hole punching. The

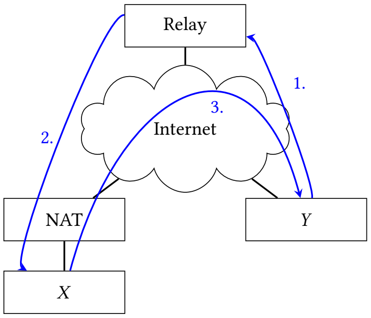

Figure 2: Connection Reversal (after [6]): Node Y connects to the relay server (1.) and asks it to relay a connection to node X (2.). After exchanging their respective addresses in the *CONNECT* message of the *DCUtR* protocol, X first dials Y directly (3.). This connection attempt only succeeds if Y is a public node.

initiator sends this message to its peer, which replies with a *CONNECT* message. The initiator uses the exchange of these messages to measure the network round trip time of the relayed connection.

The initiator responds with a *SYNC* message on the same stream. When the peer receives this message, it immediately starts dialing the initiator's addresses (as conveyed in the *CONNECT* message. The initiator waits for half the round trip time (as measured in the previous step), then it also starts establishing a direct connection. Assuming a symmetric path between both peers, this synchronization procedure leads to both nodes sending first packet to their peer at approximately the same time, thereby creating the required NAT mapping to let the peer's packet pass in.

The exact mechanism used to establish the direct connection depends on the transport protocol. In the following sections we describe how hole punching works on TCP and QUIC.

Once a direction connection between the two nodes has been established, the relayed connection is not needed anymore, and can be actively closed by the endpoints.

3.4.1 Hole Punching on TCP. TCP hole punching is a well-explored concept. It uses the fact that contrary to the normal, well-known TCP 3-way handshake (SYN, SYN-ACK, ACK), a TCP connection can also be established when both nodes receive a SYN packet (and not a SYN-ACK packet) after having sent their SYN. This is called TCP Simultaneous Open [7]. The coordination procedure described in the section above makes sure that both nodes send their respective SYN packets at the same time.

3.4.2 Hole Punching on QUIC. In order to hole punch QUIC connections, we employ a technique that – to our knowledge – has not been described in the literature before.

Contrary to TCP, where TCP Simultaneous Open is used to establish a connection, the initiation of a QUIC connection from each side would lead to two separate connections, as each QUIC connections are uniquely identified by their QUIC connection ID.

Instead we use the coordinated roles to determine the nodes' behavior: The "client" role starts dialing a QUIC connection, while the "server" sends a few UDP packets containing a random payload destined to the other node. The sole purpose of these packets is to create a NAT mapping to allow the client's packet to pass through the NAT.

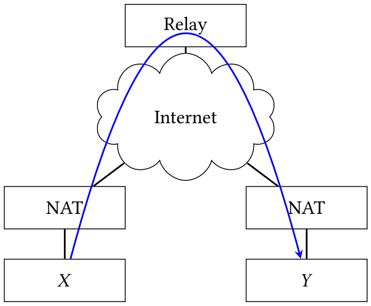

Figure 3: If both nodes are behind NATs / firewalls, the relay server is used to coordinate a hole punch through these two NATs.

4 HOLE PUNCHING IN THE WILD

4.1 Preliminary Results

During the development of the protocols, we conducted a controlled experiment with around 45 volunteers from Protocol Labs. The setup consisted of a limited relay server, a presence server, and participant nodes. Participants ran a small daemon that announced its presence and periodically tried to establish direct connections to other participants. This allowed us to test our hole punching procedures with residential networks and NAT devices.

Hole punching succeeded for 86% of the attempts on TCP and for 93% on QUIC. The vast majority of hole punching attempts succeeded on the first attempt, and we were able to show that retrying more than 3 times did not increase the success rates.

5 CONCLUSIONS

We presented *libp2p's* hole punching mechanism, establishing direct connection between peers behind NATs and/or firewalls, without reliance on centralized infrastructure. The current setup allows hole punching through NATs using endpoint-independent mappings, and can be extended to use techniques for other NAT types in the future. The hole punching mechanism has been integrated both into go-libp2p and rust-libp2p. With the v0.11.0 go-ipfs release [8], published in December 2021, Circuit v2 was enabled by default on all public nodes within the IPFS network, which equates to roughly 20% of all public nodes by the time of writing (February 2022). Once enabled by default on IPFS nodes behind NATs and/or firewalls, we plan to conduct comprehensive measurements, evaluating hole punching success across the entire heterogeneous IPFS network.

REFERENCES

[1] *libp2p.* https://www.libp2p.io.

[2] J. Rosenberg D. Wing R. Mahy P. Matthews M. Petit-Huguenin, G. Salgueiro. RFC 8489: Session Traversal Utilities for NAT (STUN). February 2020.

[3] P. Matthews J. Rosenberg T. Reddy, A. Johnston. RFC 8656: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). February 2020.

[4] J. Rosenberg A. Keranen, C. Holmberg. RFC 8445: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal. July 2018.

[5] C. Jennings F. Audet. RFC 4787: Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. January 2007.

[6] D. Kegel B. Ford, P. Srisuresh. Peer-to-peer communication across network address translators. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 13, USA, 2005. USENIX Association.

[7] DARPA Internet Program. RFC 793: Transmission Control Protocol. September 1981.

[8] *go-ipfs v0.11.0.* https://github.com/ipfs/go-ipfs/releases/tag/v0.11.0.