Proposal for Peer-to-peer chat application using Hole-punching technique

Phi H. Nguyen
Earlham College
Richmond, Indiana
phnguyen17@earlham.edu

KEYWORDS
p2p, NAT hole-punching, decentralization

2 BACKGROUND

To understand peer-to-peer networks, it is imperative we address the challenges of the network - the reasons why it is not a popular choice when compared to other architectures. The main reason, in a few words, is that our current Internet architecture is just not optimized for it[1]. Although there are many challenges in Peer-to-peer network in dealing with file transfer, this section only focuses on the challenges that a peer-to-peer chat system may encounter.

2.1 Missing data

Stutzbach et al. addresses the challenge of transferring data from one client directly to another client[5]. Since the connection relies on the bandwidth of the two users, which could differ significantly, packets can be missing during the transmission. There must be a way to handle missing data. This problem has been solved relatively well in a client-server structure, but not in a Peer-to-peer environment.

2.2 Dynamic addressing

One thing that client-server architectures have been able to solve but peer-to-peer have not is dynamic addressing. In client-server architecture, the server only sends data to the client once requested or once a session has been established. Moreover, the address of the server is constant, hence, the client will always know where to request and always expect to get an answer. For Peer-to-peer, once a client address changes, due to dynamic addressing by DHCP and PPP, any peer that does not have this new address will not know where to look[5].

2.3 Network Address Translation

Perhaps Network Address Translation is the most discussed challenge in designing a Peer-to-peer application because of the complexity as well as popularity of the problem.

Network Address Translation(NAT) resides in majority of home routers/private networks. Its job is to block unwanted outside requests to a private network and only allows authorized ones. When we want to make a request to someone else, NAT records our local IP address and port number, then sends out the request using the public IP address and a new port number that it assigns to our request. Then, when receiving the data, NAT will know what local
IP address/port number to send it to, since the public port has been opened when sending the message. Only a request to opened port can be forwarded to the recipient.

This is the main reason why Peer-to-peer cannot function well in the current Internet architecture. When a peer wants to send a message to another peer to initiate the conversation, it would not know what port is open on the other side, hence the message will be automatically dropped by the recipient’s NAT.

Not only is bypassing NAT difficult, but there are also a variety of NATs in real world that have different configurations, designed by different brands. Some methods can bypass some particular NAT models but not the others [1].

2.4 Peer reputation
Before connecting two peers, we need to determine the reputation of each peer, whether it is a trustworthy client. Trustworthiness here can involve many things: low bandwidth, low capacity, low uptime, etc. [6].

3 RELATED WORK
While there are various issues need to be solved in a peer-to-peer network, this section will focus on how to bypass Network Address Translation (NAT). Since it is one of the first and also most important parts to get a connection between two peers - also because it’s the major focus of the proposal.

3.1 Universal Plug and Play
Developed by Microsoft, Universal Plug and Play (UPnP) NAT Traversal is a technique used in Windows XP. When a new host needs a connection, a Windows XP machine configures the network addressing and broadcasts its presence in a subnet. This machine can act as a NAT Gateway, or a control point, and detect whether it is behind a NAT. This NAT Traversal technique allows peer-to-peer applications to traverse the NAT Gateway by dynamic control of public ports. The main downside of this technique is insecurity, and only a subset of routers allow this technique to be used[3].

3.2 Simple Traversal UDP Through Network Address Translators (STUN)
STUN is a lightweight protocol that lets devices behind NAT discover the type of NATs between them. A STUN client learns about the port assignment of a NAT by sending a STUN request and expects to receive a STUN response. A STUN request is a Binding Request over UDP. This Binding Request would arrive in a STUN server that sits between the clients. Since the request could traverse through multiple NATs, it carries the address of the last NAT device to the STUN server, which would send a STUN response that contains this address back to the client. By comparing the local address with the content of the response, a client can determine whether they are behind a NAT, and aware of the last NAT before it could reach another peer (assuming it is on a different subnet)[3]. The downside of this is that STUN does not work well with Symmetric NAT, which creates a binding based on the source and destination IP addresses and ports. This technique creates a new IP address after reaching the server and hence would fail the connection with other peers[3]. Moreover, STUN requires the application to be upgraded to support the public STUN server, which makes this technique very hard to deploy[3].

3.3 NAT hole-punching
NAT hole-punching might be the most popular technique to help a peer request bypass a NAT [1]. Before a request can reach the outside world, the private IP/port needs to be translated into a public IP/port via the NAT. Any response/request to this public port at this public IP address will be forwarded to the client. Hence, this port has been opened and there is a “hole” in the NAT table. Hole punching means to open a port on the NAT so that we requests that want to reach us can bypass the NAT by going through that “hole” (or public port).

Generally, hole punching would need a relay server that sits between the clients to establish a connection. This technique can use the server to maintain the connection between them or leave it out of the equation once the connection has been established.

Hu studies the architecture of BitTorrent and Skype and claimed that while a Peer-to-peer over TCP is possible, UDP is a more favored transport[3]. The reason for this is that TCP needs a three-way handshake before they can actually send data, while UDP does not require a session to be established for the data to be sent. In the untrusted and unreliable environment of Peer-to-peer network, UDP is therefore easier to manage and deploy. The architecture that Hu suggests also aligns with the architecture of a relay server, though the name was changed to “connection broker.” This architecture solves the problem posed by STUN by using an additional trick: they use the same IP/port with the brokers and with the other peers, hence it does not create a new IP/port pair in a Symmetric NAT.

Ford claims that having a relay server to forward messages is generally a more robust technique[1]. Using a hybrid architecture, the network can still rely on the client-server method while eliminating the fear of users’ data being stored in the centralized server. The role of the relay server in this scenario is merely to forward messages and nothing else. Appealing it may look, however, it is still a single point of failure architecture — the two clients need the server to be up all time to maintain a session — and the clients do not know if their data is being stored or not and whether or not they should trust this server.

Another method is to leave the relay server once the connection has been established. When A wants to connect with B, it sends a request to the response server. The data includes the public IP/port pair of A. The relay server will record A and its target client B in a table. When B connects to the server and requests to connect with A, the relay server forwards A’s IP/public port address to B and B’s ones to A. Now A and B know each other opened port, hence can communicate with each other. One of the issues that might occur is that the port must be opened before the target client sends packets. One possible solution is to confirm the connection between A to server and B to server before connecting the two together.

4 DESIGN
Generally, this design is inspired from Ford’s architecture of NAT hole punching[1]. The network needs a relay server for peer discovery and to bypass the NAT (if there is any) that the peers are
Proposal for Peer-to-peer chat application using Hole-punching technique

5 RESOURCE
I will use a free tier EC2 on AWS for the relay server, so no material cost will be made for this project. The technology stack I would be using for the relay server could be a NodeJS or a Flask server with MySQL as the database. For the p2p communication, I will use WebRTC which supports p2p communication via browser.

6 TIMELINE
Week 1 (5 Oct): Writing the first draft of report/paper, setting up environment
Week 2 (12 Oct): Writes the server logic for managing chat rooms, submit the first draft Week 3 (19 Oct): Write the p2p module for communicating via browser
Week 4 (26 Oct): Wrapping up the p2p module, submitting the second draft of paper
Week 5 (2 Nov): Refactoring code, submit the second release of software
Week 6 (9 Nov): Code freeze, review the paper
Week 7 (16 Nov): Submitting software and paper

7 CONCLUSION
In conclusion, data privacy is a significant issue in today’s centralized architecture. Peer-to-peer networks are one of the solutions to decentralize the Internet, making each connection become autonomous and anonymous. While there are many challenges to this protocol due to the way our Internet is configured, there are also proposed solutions to solve. One of the biggest challenges is NAT, which blocks unwelcoming requests from outside world to our computer. While this is beneficial in terms of security, it also blocks Peer-to-peer transmission. The paper presented multiple ways to bypass a NAT and the most popular and robust method yet is Hole-punching with a relay server.

5 RESOURCE
I will use a free tier EC2 on AWS for the relay server, so no material cost will be made for this project. The technology stack I would be using for the relay server could be a NodeJS or a Flask server with MySQL as the database. For the p2p communication, I will use WebRTC which supports p2p communication via browser.

6 TIMELINE
Week 1 (5 Oct): Writing the first draft of report/paper, setting up environment
Week 2 (12 Oct): Writes the server logic for managing chat rooms, submit the first draft
Week 3 (19 Oct): Write the p2p module for communicating via browser
Week 4 (26 Oct): Wrapping up the p2p module, submitting the second draft of paper
Week 5 (2 Nov): Refactoring code, submit the second release of software
Week 6 (9 Nov): Code freeze, review the paper
Week 7 (16 Nov): Submitting software and paper

7 CONCLUSION
In conclusion, data privacy is a significant issue in today’s centralized architecture. Peer-to-peer networks are one of the solutions to decentralize the Internet, making each connection become autonomous and anonymous. While there are many challenges to this protocol due to the way our Internet is configured, there are also proposed solutions to solve. One of the biggest challenges is NAT, which blocks unwelcoming requests from outside world to our computer. While this is beneficial in terms of security, it also blocks Peer-to-peer transmission. The paper presented multiple ways to bypass a NAT and the most popular and robust method yet is Hole-punching with a relay server.

REFERENCES