

# Implementing NAT Traversal on BitTorrent

Yi-Wen Lai  
and KuangFu Lai

Dept. of Computer Science and Information Engineering  
National Taiwan University, Taipei, Taiwan

**Abstract**—Network Address Translation (NAT) causes difficulties for peer-to-peer (P2P) communication. For BitTorrent, if only one client is behind NAT, it is possible for two clients to build connection. But for both clients are behind different NATs, currently it is not possible without specific settings on their own NAT.

In this work, we integrate TCP NAT traversal scheme into BitTorrent by modifying open source client. Our approach do not need super user privilege and only requires a few modifications.

## I. INTRODUCTION

The Internet architecture today is vastly different from that envisioned when TCP/IP was designed. Firewalls and NATs often make it impossible to establish a connection even if it does not violate policy. For traditional client/server communication, NAT is not a problem when the client is on a private network and the server has public IP. But it is difficult for two nodes on different private networks to contact each other.

There are some proposed approaches to solve NAT problem, such as relaying, connection reversal, and hole punching. Relaying is the most reliable but least efficient way to do NAT traversal. It consumes the server's processing power and network bandwidth, and usually increase communication latency. Connection reversal requires the server involved only at the connection building phase. But it has the limitation that one of the clients must have public IP. Hole punching is a simple approach to do NAT traversal. But this method depends much on the behavior of NAT. It is solved for UDP by STUN [3]. For TCP, there are more issues.

The rest of this report is organized as follows. Section 2 and Section 3 discuss the proposed TCP NAT traversal approaches and observed NAT behavior. Section 4 analyzes port-prediction and Section 5 presents our implementation. Section 6 concludes this report.

## II. TCP NAT TRAVERSAL

Some approaches using hole punching for TCP NAT traversal have been proposed in recent literature [1] [2] [4] [5]. In all the approaches, there will be a server which has a public IP to help establishing the connection. Both ends would initiate a TCP connection. The outbound SYN packet from each host creates the necessary NAT state for its own NAT. Each approach then reconciles the two TCP attempts into a single connection through different mechanisms.

Some of these approaches require super user privilege in order to observe TCP sequence number or set the value

of TTL. This requirement is not acceptable for most users. Therefore we choose the approach in [2] which only requires TCP simultaneous open (we discuss this in Implementation Section).

## III. NAT TCP CHARACTERISTICS

In [1], the authors provide a detailed survey on NAT TCP characteristics. Due to the limitation of length, we only discuss the properties which are most important factors in our experiments, NAT Mapping and Endpoint Filtering.

### A. NAT Mapping

A NAT chooses an external mapping for each TCP connection based on the source and destination IP and port. Some NATs reuse existing mappings under some conditions while others allocate new mappings every time. For UDP, it is known that some NATs assign a fixed address and port for all connections originating from a fixed source address and port [3].

We list NAT Mapping types in Table 1. The *Lab* and *Wild* columns are the observation of [1], and our observations of four NAT boxes are in the last column.

NAT Mapping	Lab[1]	Wild[1]	Our
NB:Independent	9	70.1%	2
NB:Address and Port <sub>1</sub>	3	23.5%	1
NB:Connection <sub>1</sub>	3	3.9%	0
NB:Port <sub>1</sub>	0	2.1%	0
NB:Connection <sub><math>\mathcal{R}</math></sub>	1	0.5%	1

TABLE I  
NAT MAPPING TYPES

NB:Independent means that when the source IP and port are fixed, the external mapping is also fixed no matter what the destination IP and port are. NB:Address and Port<sub>1</sub> means that when the destination IP or port is different, the external mapping will change. The subscript '1' denotes that the difference between new mappings is 1. NB:Connection<sub>1</sub> means that every connection would result a different external mapping and the difference is 1. NB:Port<sub>1</sub> is similar to NB:Address and Port<sub>1</sub> except that only different destination port would change the external mapping. NB:Connection <sub>$\mathcal{R}$</sub>  means the external mapping would change in every connection but has no pattern. The subscript ' $\mathcal{R}$ ' indicates random.

In the four NAT boxes we have, two have NB:Independent behavior (FreeBSD NAT, ShareTech Firewall), one have

NB:Address and Port<sub>1</sub> (Windows XP ICS), and one behaves random pattern (FreeBSD pf).

### B. Endpoint Filtering

End Filtering	Lab[1]	Wild[1]	Our
EF:Address and Port	12	81.9%	4
EF:Address	1	12.3%	0
EF:Independent	3	5.8%	0

TABLE II  
ENDPOINT FILTERING TYPES

NAT may filter inbound packets addressed to a port unless certain conditions are met. If no NAT mapping exists at that port, a NAT is forced to filter the packet since it cannot forward it. If a mapping exists, however, it may require the source address and/or port of the inbound packet match the destination of a preceding outbound packet.

We list the Endpoint Filtering types in Table 2. EF:Address and Port means NAT would allow the inbound packet only if the source IP and port is the same as the destination IP and port of the preceding outbound packet. EF:Address only check the IP of inbound packet, and EF:Independent would allow any inbound packet if there exists a mapping.

### IV. PORT PREDICTION

Although a large number of NATs have NB:Independent behavior, there are a unignorable portion other than NB:Independent. When a NAT does not have NB:Independent and EF:Independent behavior, both ends need the information of the other's external port mapping which is different from the port they connect to the server. Therefore, we need to do port prediction.

Port prediction involves two stages. First, when a client contacts the server at the first time, the server has to determine the type of NAT the client is using. Usually this involves connections more than once. Second, according to the type of NAT, the server needs to tell other clients which port will be used by that client when they want to establish connection to it.

The second part is not easy when there are multiple clients in a network, since we may not precisely control the order of connections between clients. Currently, we have not yet developed a robust approach to do port prediction.

### V. IMPLEMENTATION

In the four NAT boxes we have, all of them are EF:Address and Port. This means both clients need to know the correct external port of the other end's NAT in order to complete the hole punching procedure. Therefore, FreeBSD pf which has a random port behavior is not possible for TCP hole punching.

In current status, we have successfully implemented TCP NAT traversal on BitTorrent between two NATs which have NB:Independent and EF:Address and Port behavior. We use BitTornado [6] as the BitTorrent client, and Bit Tracker [7] as the BitTorrent tracker and public server. Both of these

softwares can be executed without super user privilege. There are five machines in our experiment environment, including one server, two NAT boxes, and one client behind each NAT. We label them as S (server), N1, N2 (NAT), C1, C2 (client), while C1 is behind N1 and C2 is behind N2.

The traversal scenario is quite simple, finished in three rounds. The first round, C1 contacts the server S, S records the IP and port it sees, which is the external mapping of C1. The second round, C2 contacts the server S, S also records the IP and port it sees. Then S tells C2 the information about C1. C2 will try to contact C1 using the information given by S. The packet will be blocked by N1 since the destination of preceding outbound packet is S, not N2 (or we can say the external mapping of C2). Third round, after an update interval (a time interval decided by tracker), C1 contacts S again. This time C1 gets the information of C2, and C1 will try to contact C2. Since C2 already attempted to contact C1, there is a mapping existing in N2. Therefore, C1 can establish the connection successfully. And the NAT traversal is done.

The key point in modifying BitTorrent client is to let all communication use the same port. In original BitTorrent protocol, only the listening port is fixed. Sending announce request and BitTorrent handshake use arbitrary port. Client tells tracker its listening port by the information in announce request. When client is behind NAT, the external mapping port may not be the same as the port which the client is listening. After we modified the client to always use the same port, we also need to modify the tracker to record client listening port by real connection port instead of the information in announce request. The detail of code modification can be found in [8]. There are only less than twenty lines need to be added or modified.

### VI. CONCLUSION

In this work, we have implemented TCP NAT traversal scheme into a real life application, i.e. BitTorrent. While not all existing NATs can be successfully traversed, the results are still encouraging given that only a couple years ago, it was widely assumed that TCP NAT traversal was simply not possible.

In our implementation, the client and tracker both need no super user privilege. The concept is simple that most existing BitTorrent softwares can integrate TCP NAT traversal with only a few modifications. There are more and more applications using BitTorrent technology as their network part. When clients behind NAT can contribute their upload capacity to a BitTorrent network, applications like P2P video streaming can be more promising.

### REFERENCES

- [1] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," 2005.
- [2] Bryan Ford, Pyda Srisuresh and Dan Kegel, "Peer-to-Peer Communication Across Network Address Translators," 2005.
- [3] J. Rosenberg, J. Weinberger, C. Huitema and R. Mahy, "RFC 3489: STUN-Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," 2003.

- [4] A. Biggadike, D. Ferullo, G. Wilson and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," 2005.
- [5] J. L. Eppinger, "TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem," 2005.
- [6] BitTornado. <http://www.bittornado.com/>
- [7] Bitt Tracker. <http://www.btiteam.org/>
- [8] <http://www.cmlab.csie.ntu.edu.tw/~franklai/nat.htm>