[PKS08] *Konstantin Pussep, Aleksandra Kovacevic, Ralf Steinmetz;* **On NAT Traversal in Peer-to-Peer Applications.** In: IEEE: WETICE Collaborative Peer-to-Peer Systems Workshop (COPS'08), June 2008.

On NAT Traversal in Peer-to-Peer Applications

Konstantin Pussep, Matthias Weinert, Aleksandra Kovacevic and Ralf Steinmetz
Multimedia Communications Lab, TU Darmstadt
{pussep, sandra, steinmetz}@kom.tu-darmstadt.de

Abstract

A widely used technique to overcome the shortage of unique public IP addresses is Network Address Translation (NAT), which hides several hosts behind a single public address. This method works smoothly with client-server architectures; however, it causes severe problems with the peer-to-peer (p2p) communication paradigm. Due to the side effects of NAT, the establishing connection is made possible only by using special NAT traversal techniques. This paper presents a lightweight framework for NAT traversal, which smoothly integrates with p2p applications. The framework can be easily used by most p2p applications, is extensible and does not require additional maintenance overhead.

1. Introduction

Network Address Translation (NAT) devices introduce a severe problem for peer-to-peer systems, since every peer should be able to contact the other peers in order to use their services. For example, in a Distributed Hash Table (DHT) [3, 7] the peers hidden behind NAT cannot be easily accessed by other peers and therefore cannot contribute to the DHT routing. Unfortunately, p2p applications have to deal with the majority of users being hidden behind NAT [8].

Only special NAT traversal techniques make these connections to hidden hosts possible. However, the success rate of different techniques depend on the NAT device used and the implementation is complex. Most of the NAT traversal solution are either part of proprietary applications or nested in the application code. More general approaches [2] or [1] are application domain specific or not light-weight. Other approaches, e.g. [6], [5], do not provide all the functionality required for a p2p application.

In this paper we present a simple yet extensible solution for NAT traversal for p2p applications. The solution is *simple* in the sense that it does not incur additional maintenance overhead and can be employed by any p2p application. Furthermore, it is *extensible* in that new traversal techniques can be added later on.

2. Background

Handling of NAT devices is complex because different implementations show different behavior. If a NAT device forwards an initial message from a host inside the internal network to an external host, a mapping is created that enables the device to forward later answers to the correct internal machine. Depending on the type of the NAT implementation, this mapping contains different information [4].

While such mechanisms like port forwarding and UPnP could solve the NAT traversal in a reliable way they must be provided at the user side and the developer of a p2p application cannot rely on their existence. Other techniques can be implemented by an application itself, for example, connection reversal, hole punching and relaying (as fallback). All these techniques are discussed in detail in the long version of this paper [4].

3. Framework

In addition to the low maintenance overhead, a flexible NAT traversal framework has to offer a *simple API* and be *extensible* to support an easy integration of new NAT traversal mechanisms. We assume that each instance of the client application has a globally unique identifier. If an instance cannot be reached directly, our framework uses this ID to distinguish multiple hosts with the same public IP address. For example, in a DHT each peer already has a unique ID that can be reused. Further, the framework can ask the application instance for possible relay hosts for the peer with a given ID. If the application cannot provide a potential relay peer, the framework can make use of public relay hosts. This is done to avoid maintenance overhead for an additional overlay. Many p2p applications, such as DHTs or BitTorrent already know peers able to coordinate the NAT traversal process.

Our framework is implemented in Java and make use of Java New IO. The framework architecture is shown in Figure 1. Application developers need to use the *NAT subsys-*

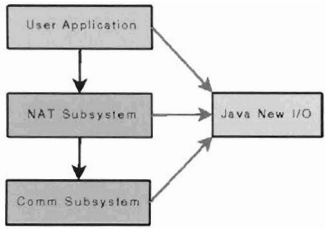

Figure 1. Framework architecture

IConnectionBroker
requestTCPChannel(PeerHandle h): SocketChannel
requestUDPChannel(PeerHandle h): DatagramChannel
registerChannel(DatagramChannel c)
registerChannel(ServerSocketChannel c)

(a) Design of the IConnectionBroker

IApplicationCallback
getOwnPeerID(): PeerID
getRelayHostsFor(PeerHandle h): DatagramChannel

(b) Design of the IApplicationCallback

Figure 2. Interfaces

tem in order to use the NAT traversal mechanisms. This subsystem provides the application with standard `java.nio` channels which are used to perform the real I/O operations. The *NAT subsystem* uses the `java.nio` and the *Communication Subsystem*. It contains the components that enable clients of the framework to establish connections with peers behind NAT devices. Finally, the *Communication Subsystem* provides functionality to establish a message-based communication between peers.

The main interface exposed to the application is the *IConnectionBroker* as presented in Figure 2(a). The first two methods allow the establishment of a channel to a given peer. This peer is specified by the *PeerHandle* interface that contains the peer's address and ID. The two *registerChannel* methods give application developers the capability to inform the framework that a given channel can be used for signaling. Such a channel can be used to coordinate NAT traversal actions or can be the target of relayed communication. Possible parameters are UDP channels and TCP server channels.

The second interface shown in Figure 2(b) is the *IApplicationCallback* which has to be implemented by the application developer. The method *getOwnPeerID()* returns the ID of the local application instance. The traffic generated by the NAT subsystem, for example punch requests, will include this ID to prevent confusion when several peers share the same public IP address. The second method requests all known peers who might act as a relay for a target peer.

NAT traversal techniques have to be implemented as special *strategies* specifying how to establish communication. In this paper, we implemented the following techniques: connection reversal, UDP hole punching, and relaying as a fallback solution. For the reasons discussed in Section 2 we skipped TCP hole punching and UPnP. Upon an application request the framework selects the most promising traversal strategy according to the NAT status of the involved peers. As several strategies may be possible, the order in which these techniques are deployed is defined by their priorities. The technique with the highest priority starts to process the request; if it fails, the next technique will be invoked. Once a valid result is returned, the connection is established and peers exchange data transparently, i.e. unaware of the NAT devices in between.

In order to evaluate our framework we developed an instant messenger with a file transfer function. The application relies on our framework for NAT traversal. We tested the NAT traversal in two fixed environments, one with virtual machines and one with real Internet hosts. In both cases some peers were hidden behind a NAT while other were publicly available. The framework proved able to find a suitable mechanism for each scenario, while in some setups (TCP connection with both peers behind different NATs) it had to resort to relaying with a third peer acting as a relay.

4. Conclusion

In this work we pointed out the connectivity problem resulting from the need to establish communication with peers behind NAT devices. We presented a lightweight solution that has been shown capable of providing an easy way to integrate NAT traversal functionality into p2p applications. Currently, we are integrating this framework in a file sharing application for collaborative communities.

References

[1] Jxta project. http://www.jxta.org.

[2] Google. Libjingle. `http://code.google.com/p/libjingle/`.

[3] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, 2002.

[4] K. Pussep, M. Weinert, N. Liebau, and R. Steinmetz. Flexible framework for nat traversal in peer-to-peer applications. Technical report, TU Darmstadt, Nov 2007.

[5] J. Rosenberg, R. Mahy, and C. Huitema. Traversal Using Relay NAT (TURN), 2005. `http://www.jdrosen.net/papers/draft-rosenberg-midcom-turn-08.txt`.

[6] J. Rosenberg et al. Stun-simple traversal of user datagram protocol (udp) through network address translators (nats). *RFC3489, March*, 2003.

[7] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[8] M. Zhang et al. Large-scale live media streaming over peer-to-peer networks through global internet. In *P2MS*, 2005.