홀펀칭 : <http://blog.bagesoft.com/608>  
  
  
출처 : <http://cafe.naver.com/cppworld.cafe>  
  
홀펀칭 관련 문/답 링크  
  
[http://www.devpia.com/maeul/contents/de ··· 3D613698](http://www.devpia.com/MAEUL/Contents/Detail.aspx?BoardID=50&MAEULNo=20&no=613698&ref=613698)

UDP에서는 socket API를 호출한다고 해서 바로 LocalPort가 binding되지 않으며 sendto를 호출한 이후에 비로소 binding 됩니다.

따라서, UDP에서 LocalPort 번호를 얻기 위해서는

1. sendto를 호출한 바로 직후

2. recvfrom이 정상적인 값(SOCKET\_ERROR이 아닌 값)인 경우

인 경우에 LocalPort를 알 수 있습니다.

LocalPort를 얻기 위해 사용되는 모듈 첨부합니다.

[TCP 에서 local port 구하는 방법]

int CVDTCPSession::GetLocalPort()

{

    int nSize;

    int nRes;

    nRes = 0;

    if (m\_nSocketHandle == INVALID\_SOCKET) goto \_end;

    nSize = sizeof(m\_sAddr);

    if (getsockname(m\_nSocketHandle, (sockaddr\*)&m\_sAddr, &nSize) == 0)

    {

        nRes = ntohs(m\_sAddr.sin\_port);

    }

\_end:

    return nRes;

}

[UDP 에서 local port 구하는 방법] (sockaddr\_in 객체를 복사해서 사용해야 합니다. 원래의 값을 잃어 버리지 않기 위해서입니다)

int CVDUDPSession::GetLocalPort()

{

    int nSize;

    sockaddr\_in sAddr;

    int nRes;

    nRes = 0;

    if (m\_nSocketHandle == INVALID\_SOCKET) goto \_end;

    nSize = sizeof(sAddr);

    sAddr = m\_sAddr;

    getsockname(m\_nSocketHandle, (sockaddr\*)&sAddr, &nSize); // do not check result

    nRes = ntohs(sAddr.sin\_port);

\_end:

    return nRes;

}

Hole Punching... 일명 NAT 구멍뚫기라고 하죠. ㅎㅎ

원래 용어는 STUN인데, 국내에서는 Hole Punching으로 알려 진 것 같더군요.

UDP 기반의 P2P 서비스를 구축하는 데에는 STUN을 꼭 알아야 하죠.

몇년전만 하더라도 별로 알려지지 않았었는데, 요즘은 많이 보급이 된 것 같습니다.

참고로 알려 드릴 사항은 Hole Punching을 구현한다고 해서 모든 NAT에서 작동되지는 않는다는 겁니다.

STUN의 종류는 다음과 같습니다.

1. Full Cone

2. Restricted Cone

3. Port Restricted Cone.

4. Symmetric NAT

우리가 일반적으로 집에서 사용하는 공유기는 Full Cone 타입이고

회사나 대규모 업체에 가면 Symmetric NAT의 타입을 많이 사용합니다.

(수천, 수만대의 컴퓨터를 내부 IP로 사용하기 위해서는 Symmetric NAT 도입이 불가피함)

Cone이라고 하는 것은 우리가 수퍼마켓에서 사 먹는 "콘"이라고 보시면 되구요,

쉽게 깔때기 형태라고 보시면 됩니다.

깔때기의 주둥이가 넓으면 넓을 수록 바깥에서 패킷을 쐈을 때 쉽게 깔때기 주둥이로 골인될 수 있겠죠? ^^

깔때기의 주둥이가 제일 넓은 게 Full Cone이구요,

깔때기의 주둥이가 제일 좁은 게 Symmetric NAT 입니다.

STUN에 대한 자세한 사항은 STUN에 대한 문서를 참고하시기 바랍니다.

http://www.gilgil.co.kr/bbs/view.php?id=lecture&page=2&sn1=&divpage=1&sn=off&ss=on&sc=on&select\_arrange=headnum&desc=asc&no=144

불행한 사실은 만약, 양쪽이 모두 Symmertic NAT 환경이라면 Hole Punching이 적용되지 않습니다.

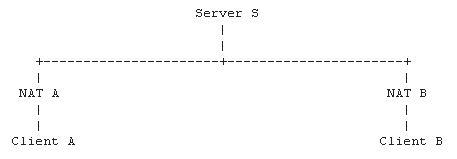
이를 위해서 상대방의 Local Port(호스트의 Local Port가 아닌 NAT 장비의 Local Port)를 추축(guessing)하는 기법이 있기는 한데

아직 안정화 단계는 아닌 것 같습니다.

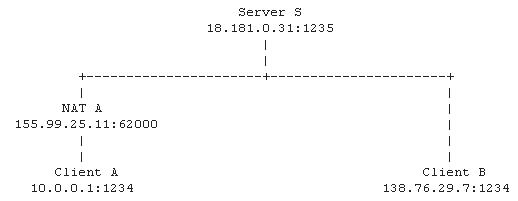
**[출처]** [홀펀칭관련 - 내부 사설 IP 에 대해서 ..](http://blog.naver.com/ta4rom/20032952710) |**작성자** [따닷](http://blog.naver.com/ta4rom)

|  |
| --- |
|  |

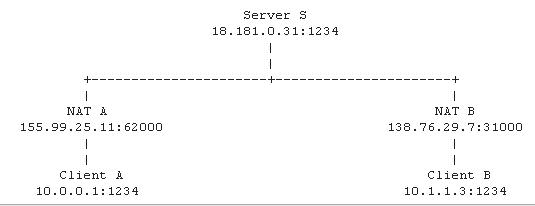
네트워크 응용 수업의 최종과제가 홀펀칭(Hole Punching) 기술로 구현한 에코 클라이언트의 구현이라,  
게임프로그래밍이나 여러 P2P 프로그래밍에서 폭넓게 사용되는 기술이라고 한다.  
일단은 프로그래밍을 해야 하는데 생소한 정보들이여서 몇몇 관련문서를 보고 이해를 하려고 노력해 보았다.  
  
**0. 들어가기 전 알아두어야 할 용어들.**  
- NAT [Network Address Translation]  
OSI 모델의 3계층인 네트워크 계층에서 사설 IP 주소를 공인 IP 주소로 변환하는데 사용하는 통신망의 주소 변환기.  
<http://100.naver.com/100.nhn?docid=717874>  
우리가 흔히 말하는 공유기, 라우터 등이 이런 역할을 수행한다고 보면 된다.  
  
**1. 개요**  
Hole Punch 란 종이에 철을 하기 위해 동그란 구멍을 뚫어주는 사무용 기기이다.  
그런 개념으로 접근하여 막힌 뭔가를 뻥 뚫어준다는 것으로 이해하고 출발하여 보자.  
일단은 P2P 연결을 위한 여러 기술들을 알아보기로 한다. (홀 펀칭은 그 중의 하나이므로..)  
  
  
**2. 이해**  
- 전제 : 통신을 하기 위한 Client A와 Client B가 존재하며경우에 따라 NAT아래 놓일 수 있다.  
이들의 통신을 중계하기 위한 Server S가 존재한다.  
NAT과의 P2P 커뮤니케이션하기 위해서는 몇가지의 기술이 있다.  
Relaying, Connection Reversal, UDP Hole Punching, UDP Port Number Prediction, Simultaneous TCP Connection Initiation 이 그것이다.  
일단 Relaying부터 살펴보도록 하자.  
  
**3. 기술**  
**3.1 Relaying**



Relaying은 간단히 중계 서버를 두고 중계 서버를 통해 통신을 한다.  
A가 서버 S에게 메세지를 보내고 서버 S는 B에게 메세지를 보낸다.  
A와 B가 서버에 접속이 유효한 동안은 메세지를 계속 주고 받을 수 있다.  
하지만 불필요한 대역폭의 낭비와 서버의 리소스를 소모하게 된다.  
  
  
  
**3.2 Connection Reversal**



이 경우는 하나의 클라이언트가 NAT뒤에 위치해 있는 경우이다.  
B가 A와의 연결을 하고자 할때 A믜 사설IP로는 당연히 접속이 불가능하고, 서버 S가 관찰하는 NAT의 공인 IP로의 접속은 A에서 나가는 것만 허용하기 때문에 역시 접속이 불가능하다.  
그래서 중계 서버 S를 이용해 "내가 못가니까 니가 나와라" 고 해서 B의 공인 IP를 서버 S에게 알려 준 다음 역으로 A가 접속을 시도하여 커넥션을 맺게 하는 방식이다.  
  
**3.3 UDP Hole Punching**



이 경우는 두 클라이언트가 전부 NAT 뒤에 있는 경우이다.  
여기서 두 클라이언트가 같은 NAT 뒤에 있는가 혹은 다른 NAT 뒤에 있는가로 다시 경우가 나뉘어 진다.  
  
  
**3.3.1 두 클라이언트가 다른 NAT 뒤에 있을 때.**  
  
클라이언트 A가 B와 커넥션을 맺고 싶을 떄 일반적으로는 3.2의 경우와 같이 커넥션이 맺어지지 않으므로 중계서버 S를 이용하여야 한다.  
서버 S에 접속할때 서버는 클라이언트 A의 정보 (사설,공인 IP)를 취득하여 저장하게 되고 B가 접속할때도 이 정보를 저장하게 된다.  
그리고 만약 A가 B에 대해서 연결을 요청하게 되면 중계서버 S는 A와 B에게 동시에 서로의 IP 정보를 보내 주게 되고 커넥션 희망여부를 전달받은 각 클라이언트는 각각 연결 시도를 하게 되어 커넥션이 맺어질 수 있게 된다.

- 추가사항 : UDP 커넥션의 경우에는 지속적인 핑퐁 메세지가 없으면 연결이 끊어질수 있기 때문에 일정간격으로 연락을 해 주어야 한다. NAT의 종류에 따라서 되는것도 있고 안되는것도 있다고 한다..(성공 확률 분포 등에 대한 자료는 PDF문서 참조)  
  
일단 기본적인 기술은 여기까지 이해하고 나머지 추가적인 기술과 내용들은 아래의 주소에서 더 확인이 가능하다.  
(사실 이거 이해를 하려고 꽤나 머리를 싸맸다. 아직 확실한지는 잘 모르겠지만서도 -\_-)  
  
[ [Network Address Translation and Peer-to-Peer Applications (NATP2P)](http://www.serious-code.net/moin.cgi/Scrap_2fNatAndP2pApplications) ]   
[ (PDF) [TCP Connections for P2P Apps](http://reports-archive.adm.cs.cmu.edu/anon/isri2005/CMU-ISRI-05-104.pdf) ]  
[ (PDF) [Peer-to-Peer Communication Across Network Address Translators](http://pdos.csail.mit.edu/papers/p2pnat.pdf) ]  
[ [P2P에서 UDP를 쓰는 이유](http://www.gpgstudy.com/forum/viewtopic.php?t=7421) (Gpg study forum) ]

이건 다른 내용 정리한 것이다 ㅋㅋㅋ

UDP hole punching이란 서로다른 NAT로 막혀있는 두 peer들 사이에 p2p 통신을 하기 위한 방법이다.

 처음에 UDP 통신용 커넥션을 열기 위해서 public IP를 사용하는 중계 서버를 이용한다.

 자세한 사항은 아래의 두 링크를 참조하자.

[1] <http://en.wikipedia.org/wiki/UDP_hole_punching>

[2] <http://gnunet.org/papers/nat.pdf>

[2]의 내용 중 중요 사항 정리

(Step1) 랑데뷰 서버로 접속

 랑데뷰 서버 S로 클라이언트 A, 클라이언트 B가 모두 접속한다.

 접속시에 클라이언트는 자신이 접속에 사용한 자신의 IP와 Port를 같이 전송해 준다.

 S는 클라이언트가 보내온 (IP, Port)와 더불에 S가 관찰한 클라이언트의 (IP, port)를 같이 기록해 둔다.

 S가 관찰한 (IP, Port)라는 것은 IP protocol packet에 나타난 정보를 기준으로한 (IP, Port)이다.

 전자를 private end point 라고 하고 후자를 public end point이라 할때, 만약 전송 경로중 NAT가 존재하지 않는다면 'private = public' 이 된다.

 이 과정에서 주의할 점은, 일부 NAT의 경우 IP packet 내부의 데이터를 분석해서 IP와 같은 것이 있으면 같이 변환 해버리는 경우가 있기 때문에, 직접적으로 IP를 싣는 것이 아니라 변형해서 실어 보내야 한다는 것이다.

 간단한 방법으로는 1's complement등으로 변경해 보내면 된다.

 (Step2) P2P UDP 세션 열기

 클라이언트 A에서 B로 연결한다고 가정한다.

(1) A는 S에게 B로의 연결을 위한 정보를 묻는다.

(2) S는 A에게 B의 public/private end point를 보낸다. 동시에, S는 B에게도 A의 public/private end point 정보를 보낸다. 이 정보를 받으면 A, B는 서로의 public/private end point를 아는 상태가 된다.

(3) A가 B의 정보를 받은 후, A는 public/private end point에게 모두 UDP packet을 보내기 시작한다. A는 응답을 지켜보다 합당한 응답이 오는 end point 하나로 전송 경로를 고정한다. 마찬가지로 B에서도 S로부터 end point 정보를 받는 순간 부터 양쪽으로 packet을 보내고, 적절한 응답이 오는 것으로 전송 경로를 고정한다.

 이와 같은 과정으로 UDP 세션을 여는 것은 대부분의 환경에서 가능하다. 같은 NAT 하에 있는 두 클라이언트, 다른 NAT 환경에 있는 두 클라이언트에서는 문제 없이 동작한다. 하지만 2단계의 NAT를 갖는 환경에서는 전송 경로를 열 수 없기 때문에 서버를 통한 통신을 해야 한다.

 UDP hole punching에 의해서 만들어진 UDP 세션을 유지하기 위해서는 주기적인 keep alive message가 필요하다. 왜냐하면 대부분의 NAT에서 UDP 세션의 hole 유지를 위해서 timer를 유지하기 때문이다. 하지만 이런 timer를 위한 표준적인 time out 시간이 없기 때문에 적절할 시간을 골라 써야 한다. 약 20초 정도가 가장 짧은 시간이라고 하니 적절히 사용하면 될 듯 하다.

 실제 인터넷 상에서 어떻게 동작하는지 살펴봤을 때, 약 82%의 경우 성공적인 동작이 되었다고 한다.