UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

Redes de Computadores I

"NAT Traversal"

Nombres	**Rol**
Pablo Álvarez	2921037-3
Hugo Herrera	2921020-9
Enzo Rivera	2921028-4
Pascal Sigel	2921055-1
Profesor: Agustín González	
Ayudante: Daniel Cárdenas	
Fecha: 28 de Julio de 2014	

RESUMEN

El siguiente documento se contextualiza en la capa de red. Se presenta, primeramente, como se le da solución al problema de disponibilidad de direcciones IP's del protocolo IPv4 basándose en la utilización de NAT's. Se plantea, a su vez, los conflictos que se generan por el uso de NAT's y cómo se logra dar respuesta mediante técnicas denominadas NAT Traversal para solucionar el problema que ocurre al tratar de establecer una comunicación entre hosts detrás de un enrutador basado en NAT. Se ejemplifican ciertas técnicas de NAT Traversal, y se da a entender el mecanismo detrás de estas técnicas que pese a no estar estandarizadas siguen todas, generalmente, una misma lógica. Finalmente se muestra la parte práctica la que consiste en un Chat que comunica, mediante técnicas de NAT Traversal, dos hosts que están detrás de routers basados en NAT.

INTRODUCCIÓN

En el contexto de la capa de red, resulta lógico pensar que para entablar una comunicación entre dos hosts es necesario que cada host tenga una identificación, denominada dirección IP, descrita en el encabezado del datagrama enviado por el emisor para llegar a destino. A su vez resulta evidente que cada IP sea distinta de las otras para el correcto envió de la información, pero estas son un número determinado de direcciones (2^{32}) lo cual las hace agotables. Se estima que las direcciones IPs disponibles en la reserva global de IANA pertenecientes al protocolo IPv4 se agotan oficialmente el 2011, como medida para solucionar en parte éste hecho se crea el mecanismo de las tablas NAT, sistema que se implementa en los routers para poder enmascarar IP's privadas, de modo que una sola IP publica sirviese para varios equipos (una subred), esto provocó un gran alivio en el agotamiento de IPv4, pues las empresas, universidades o cualquier red con varios equipos conectados se lograron encapsular bajo una sola IP, gracias a esto el nuevo protocolo de IP's (IPv6), pudo demorarse, y hasta el día de hoy, todavía se utiliza IPv4, sin embargo se espera que con IPv6 no vuelva a ocurrir el agotamiento de IP's. Si bien la implementación de tablas NAT en los routers sirvio bastante, no fueron del todo positivas en un

principio, pues el problema principal es que al enmascarar los equipos con IP's privadas, éstos no son visibles y por lo tanto no son alcanzables desde la internet, obstaculizando conexiones por ejemplo del tipo P2P, en donde un equipo cualquiera debe convertirse en un servidor, y por lo mismo debe ser visible desde el exterior de la subred.

NAT (NETWORK ADDRESS TRANSLATION)

NAT es la solución planteada al problema del agotamiento de direcciones IP's el cual consiste en entregar solo una IP pública a cada router NAT encargado de conectar una sub-red con el exterior de internet (de ahí su nombre de "pública"), y es este router el encargado de dar las IP's privadas a cada host conectado a él, éstas IP's deben ser diferentes entré si dentro de la sub-red pero es indiferente si es distinta o igual a otra IP dentro de otra subred, ya que ambas serían privadas.

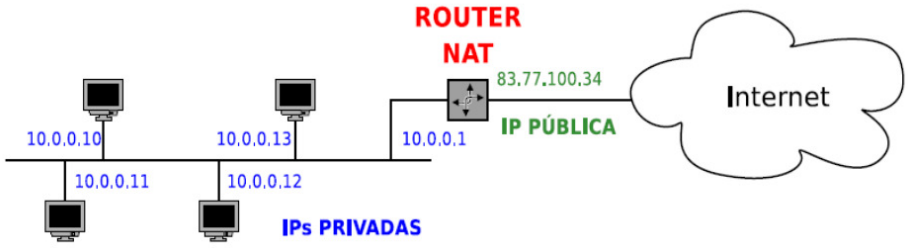

Figura 1. Esquema de Arquitectura NAT

1.1 ¿Cómo se logra entregar la información al host específico dentro de la sub-red si se desconoce su IP Privada?

Dentro del router se establece una tabla NAT la cual genera un mecanismo de trasformación entre las IP's públicas y privadas, además contiene los puertos usados tanto por el host como por el router para hacer el traspaso de información, es así como diferencia los distintos host, asignando puertos asociados a cada host en la tabla NAT como se observa en la siguiente figura:

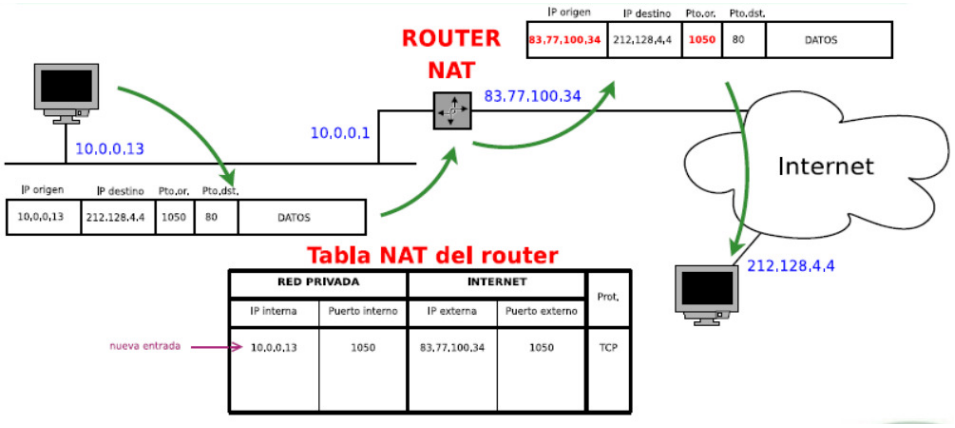

Figura 2. Funcionamiento de las tablas NAT

Es de notar que el puerto interno (del host) es el mismo que el del puerto externo (del router NAT), si otro host utiliza el mismo puerto interno que el host de IP privada 10.0.0.13, en este caso el puerto 1050, el router le asigna un puerto externo distinto para poder logar diferenciar los distintos host dependiendo de sus puertos utilizados. La IP externa siempre será la misma para cualquier aplicación puerto de cualquier host.

PROBLEMÁTICA NAT

NAT logra que un host pueda comunicarse con algún otro host de IP pública en el exterior de su sub-red y que un host de IP privada de comunique con un host dentro de una sub-red. Pero a veces se requiere que la comunicación sea entre dos host con IP's privadas dentro de dos sub-redes distintas ya sea para compartir archivos, para streaming, etc., como lo hace el protocolo P2P (Peer-to-Peer) o VoIP, entre otros. Esta comunicación no es posible a simple vista con el uso de NAT ya que ninguno de los usuarios a priori la IP privada y puerto utilizado por el otro usuario.

NAT TRAVERSAL

Como se mencionó anteriormente a veces se requiere comunicación entres IP's privadas esto se logra mediante técnicas NAT Traversal. A toda técnica que logre atravesar la barrera de los router NAT se les denomina NAT Traversal, sin importar la forma para lograrlo, es por eso que estás técnicas no están estandarizadas, pese a esto hay dos métodos que se diferencian:

1) ***Uso de un servidor como vía de la comunicación:***

   Lo que se hace en esta situación es generar dos conexiones, una entre el cliente 1 y el servidor, y la otra entre el servidor y el cliente 2, este servidor contiene una IP pública conocida a priori por ambos clientes. Primeramente ambos clientes hacen un handshake con el servidor, determinando así los puertos dentro del servidor a ser utilizados. Es así como se genera una "falsa comunicación" ya que el cliente 1 envía información al servidor, este la almacena y luego este mediante la aplicación la envía al cliente 2.

2) ***Uso de un servidor para establecer la comunicación:***

   En este caso el servidor solo es utilizado como pasarela de información. Primeramente se establecen las conexiones respectivas entre ambos clientes y el servidor, y luego este les comunica el puerto e IP públicas de los clientes a los otros clientes, encaminando así los paquetes. Cabe mencionar que no se pierde el contacto con el servidor ya que este constantemente tiene que estar informando a los clientes sobre los estados de las conexiones y puertos de los otros clientes.

Se presenta el esquema de la arquitectura NAT Traversal en general:

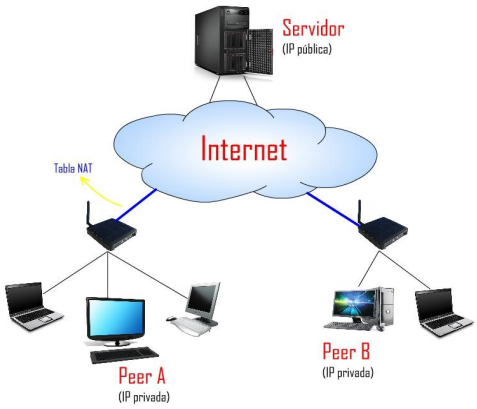

Figura 3. Esquema NAT Traversal

Acontinuación se presenta un tipo de NAT Traversal:

HOLE PUNCHING

Hole Punching es una técnica para el establecimiento de comunicaciones entre dos partes en organizaciones separadas que están detrás de firewalls. Se utiliza para aplicaciones como juegos en línea, P2P y VoIP , ambos clientes establecen una conexión con un servidor sin restricciones donde se puede obtener información de la dirección externa e interna para ellos. Dado que cada cliente inició una conexión al servidor, el servidor conoce sus direcciones IP y números de puerto asignados para esa sesión, que comparte entre ambos. Tener los números de puerto válidos hace que los servidores de seguridad acepten los paquetes entrantes desde cada lado. ICMP Hole Punching, UDP Hole Punching y TCP Hole Punching respectivamente utilizan Internet Control Message (ICM), User Datagram (UDP) y Transmission Control Protocols (TCP).

RESULTADOS DE LA PARTE PRÁCTICA

Como resultado del trabajo realizado se creó un chat por terminal el cual traspasa los NAT a través del uso constante del servidor central, con flujo de información en ambos sentidos.
En la siguiente figura se ilustra el tipo de NAT Traversal que fue usado.

Figura 4. Estructura del NAT Traversal utilizado con el flujo de información en un sentido

La idea es que el servidor central tiene IP pública por lo que los usuarios pueden abrir un NAT para la comunicación con él y luego el servidor hace de intermediario entre ambos terminales.

Para un mejor entendimiento del resultado se mostrarán los pasos a seguir que fueron efectuados en la prueba realizada:

1) Ejecutar el programa servidor en una máquina con IP pública.

$ ./serv

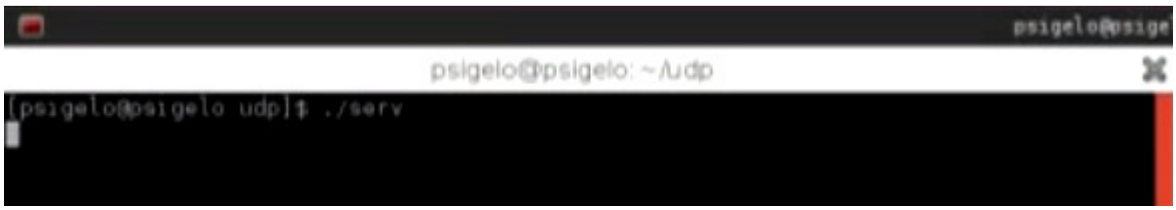

Figura 5. Ejecución del programa del servidor.

2) Usuario terminal 1 se conecta al servidor con el puerto 32000 y el usuario terminal 2 con el 32001 (así está programado el chat de prueba), la IP del servidor es 186.36.31.211.

Usuario 1:

./client_1 186.36.31.211 32000

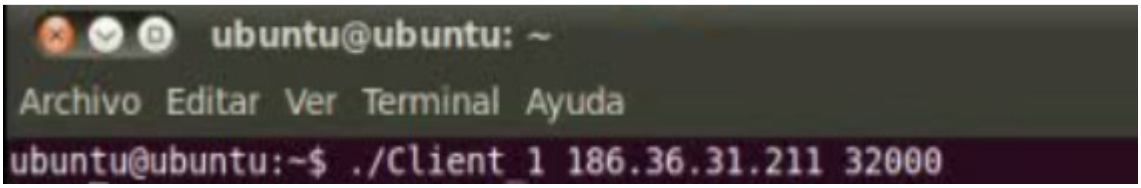

Figura 6. Ejecución del Usuario 1

Usuario 2:

./client_2 186.36.31.211 32001

Figura 7. Ejecución del Usuario 2.

Notar que para el caso del experimento, y para que fueran 3 máquinas diferentes (no usar el servidor como servidor y cliente) se usó a Aragorn como el usuario 2 por lo que se hizo una conexión SSH (conexión segura) a Aragorn y luego se ejecutó el programa dentro de Aragorn cómo se ilustra en la figura anterior.

3) Los usuarios pueden chatear libremente como se muestra en la siguiente figura.

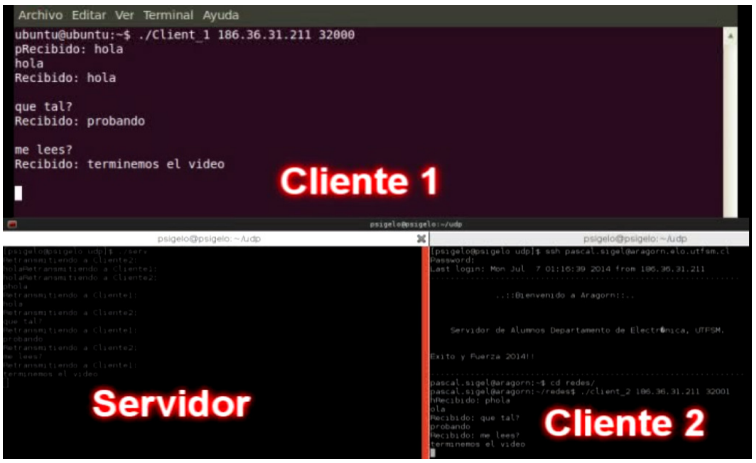

Figura 8. Chat a través de NAT Traversal.

Se adjunta en la sección anexos el código programado en C, tanto del servidor como de los clientes.

CONCLUSIONES

La necesidad del NAT es fundamental a la hora de mantener segura una IP Privada del host que se utilice, pero a la hora de querer comunicarse con otros computadores privados de otra sub-red fuera de la nuestra, es donde se encuentran obstáculos. Pero gracias a las distintas técnicas encontradas durante el trabajo, como por ejemplo NAT Hole-Punching, nos hace comprender la importancia de, algunos casos, mantener conocida la IP Privada, con el fin de compartir información desde el otro lado del mundo. Gracias a la demostración, podemos ver el potencial extrapolado que muestra; imagínense querer utilizar un robot ubicado en china, que trabaja tanto servidor como también cliente, y desde Chile, se le envía información para utilizarlo, podemos comunicarnos con él para que él nos genere respuestas y nos comparta esa información para luego esperar por acciones a ejecutar por parte de nosotros.

REFERENCIAS

(1) http://www.exa.unicen.edu.ar/catedras/comdat1/material/NAT.pdf

(2) HTTP://NZCSRSC08.CANTERBURY.AC.NZ/SITE/PROCEEDINGS/INDIVIDUAL_PAPERS/PG242_NAT_TRAVERSAL_TECHNIQUES_IN_PEER-TO-PEER_NETWORKS.PDF

(3) HTTP://GROTHOFF.ORG/CHRISTIAN/PWNAT.PDF

(4) HTTP://ALLIANCE.SEAS.UPENN.EDU/TCOM502-WIKI/UPLOADS/SITE/SIPNATTRAVERSAL.PDF

(5) HTTP://PROFESORES.ELO.UTFSM.CL/~AGV/ELO322/1S14/LECTURES/NETWORK_4.4.PDF

(6) HTTP://EN.WIKIPEDIA.ORG/WIKI/TCP_HOLE_PUNCHING

ANEXOS

Se adjunta los códigos en C utilizados en la parte práctica:

a) Servidor:

```
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <poll.h>
#include <stdlib.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>

#define TIME_OUT_POLL_MSEC 100


int main(int argc, char**argv)
{
  int sockfd,n;
  struct sockaddr_in servaddr,cliaddr;
  socklen_t len;
  char mesg[1000];
  char mesg2[1000];


  sockfd=socket(AF_INET,SOCK_DGRAM,0);

  bzero(&servaddr,sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
  servaddr.sin_port=htons(32000);
  bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr));


  int sockfd2;
  struct sockaddr_in servaddr2,cliaddr2;
  socklen_t len2;

  sockfd2 = socket(AF_INET,SOCK_DGRAM,0);
```

```
bzero(&servaddr2,sizeof(servaddr2));
servaddr2.sin_family = AF_INET;
servaddr2.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr2.sin_port=htons(32001);
bind(sockfd2,(struct sockaddr *)&servaddr2,sizeof(servaddr2));



struct pollfd fd_sockfd;
fd_sockfd.fd = sockfd;
fd_sockfd.events = POLLIN;


struct pollfd fd_sockfd2;
fd_sockfd2.fd = sockfd2;
fd_sockfd2.events = POLLIN;


while(1)
{
  if(poll(&fd_sockfd,1,TIME_OUT_POLL_MSEC)){
     len = sizeof(cliaddr);
     n = recvfrom(sockfd,mesg,1000,0,(struct sockaddr *)&cliaddr,&len);
     sendto(sockfd2,mesg,n,0,(struct sockaddr *)&cliaddr2,sizeof(cliaddr2));
    // printf(" %lu : %hu \n", cliaddr.sin_addr, ntohs(cliaddr.sin_port));
     mesg[n] = 0;
     printf("Retransmitiendo a Cliente2:\n");
     printf("%s",mesg);
  }


  if(poll(&fd_sockfd2,1,TIME_OUT_POLL_MSEC)){
     len = sizeof(cliaddr2);
     n = recvfrom(sockfd2,mesg2,1000,0,(struct sockaddr *)&cliaddr2,&len);
     sendto(sockfd,mesg2,n,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
     mesg2[n] = 0;
     printf("Retransmitiendo a Cliente1:\n");
     printf("%s",mesg2);
  }

}
 }
```

b) Cliente:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
```

```
#include <strings.h>
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <poll.h>
#include <stdlib.h>
#include <netdb.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>


#define TIME_OUT_POLL_MSEC 100

int main(int argc, char**argv)
{
  int sockfd,n;
  struct sockaddr_in servaddr,cliaddr;
  char sendline[]="hola";
  char recvline[1000];
  int pid;

  if (argc != 3 )
  {
    printf("usage:  udpcli <IP address>   puerto\n");
    exit(1);
  }

  sockfd=socket(AF_INET,SOCK_DGRAM,0);

  bzero(&servaddr,sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  //servaddr.sin_addr.s_addr=inet_addr(argv[1]);
  inet_aton(argv[1], &servaddr.sin_addr);
  servaddr.sin_port=htons(atoi(argv[2]));


  sendto(sockfd,sendline,strlen(sendline),0,
         (struct sockaddr *)&servaddr,sizeof(servaddr));

  //while (fgets(sendline, 10000,stdin) != NULL)


  struct pollfd fd_sockfd;
  fd_sockfd.fd = sockfd;
  fd_sockfd.events = POLLIN;
```

```
  pid=fork();
  if(pid<0){perror("Error en fork()");exit(1);}

  if(pid == 0){ // Hijo
    // Aca se esperara por la entrada de datos
    while (1)
    {
      if(fgets(sendline, 10000,stdin) != NULL){
        sendto(sockfd,sendline,strlen(sendline),0,
            (struct sockaddr *)&servaddr,sizeof(servaddr));
      }
    }
  }

  else{// Padre
    //Aca se leeran los datos provenientes del servidor.
    while(1){
      if(poll(&fd_sockfd,1,TIME_OUT_POLL_MSEC)){
        n=recvfrom(sockfd,recvline,10000,0,NULL,NULL);
        recvline[n]=0;
        printf("Recibido: ");
        //fputs(recvline,stdout);
        printf("%s\n",recvline);
      }
    }
  }

/*
  while(1)
  {
   // sendto(sockfd,sendline,strlen(sendline),0,
    //      (struct sockaddr *)&servaddr,sizeof(servaddr));
    n=recvfrom(sockfd,recvline,10000,0,NULL,NULL);
    recvline[n]=0;
    fputs(recvline,stdout);
  }
  */
}
```