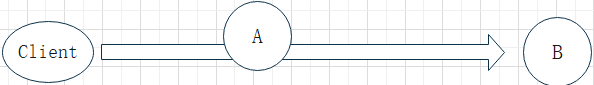
### 代理\_端口转发\_端口映射\_NAT\_概念释疑2

NAT网络地址转换：这个概念最早用于路由器和防火墙等实体网络设备上，表示路由器/防火墙设备将收到的某数据包的ip地址/传输层端口号进行修改，再走路由转发到目标主机。

NAT最初只是指network address的转换，即ip地址，后来也可以修改传输层的端口号，涉及到端口号的修改的操作也叫作PAT（port address转换），但我们仍习惯于把这二者的修改统称为NAT。 NAT是分2个方向的，对源ip:port的修改叫作sNAT（源地址转换），对目标ip:port的修改叫作dNAT（目的地址转换）

**①sNAT （源NAT）**



\*sNAT是指客户端发来的数据报文是给B的，但在路由上经过了A设备，

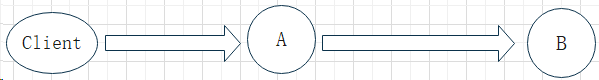
\*然后A对这个流量的源ip:port做了修改，改成了A的ip:port，并且A记录了这个转换的映射关系，

\*B收到数据后，响应的报文是发回给A，因为从B那里看，数据的源ip是A

\*A收到B发回的响应报文后，再根据 之前保存的映射关系，把响应报文的目的ip:port改为Client

\*最后响应报文走路由层 发到Client

**②dNAT （目的NAT、端口映射）**



\*dNAT如果强调对 端口的修改，也可叫作 **端口映射**

\*dNAT是指客户端发来的数据报文是给A的，但A把目的ip:port改为了B的，并记录这个转换的映射关系

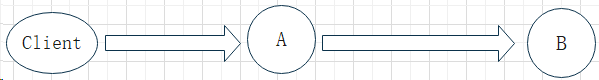
\*B收到数据后，响应的报文是发给Client的，因为之前A并没有修改报文的源ip:port，所以从B那里看，报文就是从Client发来的。

\*B发回的响应报文经路由层转发，如果再经过A时，A再根据之前保存的映射关系，把响应报文的源ip:port改回A自己的

\*响应报文再走路由层发到Client

\*\*\*注意，如果B发回的响应报文经路由转发后，不经过A设备，则源ip:port得不到修改，这样即使Client收到了响应报文，Client也不认这个报文，因为在tcp连接里，要求源ip:port和目的ip:port对得上，不然就不是同一个连接了。

**③端口转发 （port-forward）**



\*端口转发是指客户端发来的数据报文是给A的，但A把源ip:port改为A的，同时也把目的ip:port改为B的，并记录这个映射关系

\*B收到数据后，响应的报文是发给A的，因为从B那里看，收到的报文源ip:port是A的

\*A收到响应报文后，再根据映射关系，修改响应报文的源ip:port为自己，目的ip:port为Client，

\*响应报文再走路由层发到Client

\*\*\*端口转发结合了sNat和dNat，确保了B的响应报文一定发回给A，这样就能根据映射关系修改响应报文的ip:port，使Client收到正确的报文

**先来个小结吧**：

以上三种情况（sNAT、dNAT、端口转发）都只是在路由层面上对数据包的ip:port进行修改，并维护一个映射关系表，进行数据修改的设备A 本身并不处理tcp连接的问题，关于tcp等相关的事宜是由真正的Client和目标B来处理。A设备上是要求开启路由转发的功能。A的操作系统里的普通进程并不和这些转发的数据打交道，比如转发的如果是tcp层的数据，则在A的系统里用netstat命令看不到tcp连接情况

以上三种情况一般分别用于以下场景：

sNAT：用于家庭用户的出口网关，因为随着ipv4资源的耗尽，以及网民数量的增加，不可 能给每个用户分配一个公网ip，所以给用户分配的是内网ip，然后内网数据发往公网时， 要经过出口网关，这时出口网关就是上面例子中的A设备，把源ip:port进行修改，改 为A的公网ip:port，使得数据能在公网中传输。

dNAT：处于内网的用户可能有一些服务器，要给公网上的用户提供服务，但他们没有公网 ip，或者只有少数几个公网ip，所以得充分利用这些公网ip资源，对目标ip:port进行 修改，映射到内网的某服务器的某端口上，也就是访问同一外网ip的不同port，映射 到不同的内网服务器上。

**端口转发**:适用于隐藏客户端或服务端的真实ip，使用某个中介进行转发，以突破各方面对 ip的限制。

根据报文的源ip:port及目标ip:port的修改情况 可分为3种情况：

1.**只修改源ip:port**，即sNAT源地址转换，其中ip:port可以只改ip，也必须改ip

响应报文一定原路返回

2.**只修改目标ip:port**，即dNAT目的地址转换，其中ip:port可以只改其一，必改其一

目标的响应报文不一定原路返回，所以Client那边会得不到正确的响应，因为报文如果

不经过中间的A（修改报文ip:port的始作俑者），则返回时的报文ip:port得不到修改，

所以client认为此响应报文不是正确的，所以要确保响应报文在路由上能经过A

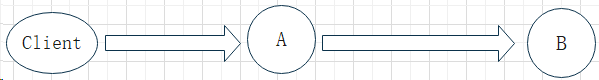
3.**修改源ip:port以及修改目标ip:port**，即端口转发，源中的ip:port可以只改ip，也必改ip

目标ip:port可以只改其一，即改ip或port都行，必改其一

响应报文一定原路返回

**④端口代理 （port-proxy）**

端口代理的用途同端口转发，但实现原理不同



\*Client访问A的某TCP端口，建立了tcp连接，然后A再和B建立tcp连接，这2个连接也 是做了映射关系的记录的

\*Client向A发送的tcp载荷数据，由A再发给B

\*B响应的数据是发给A的，A收到后再把响应数据发给Client

和端口转发不同的是：

\*A转发的只有载荷数据，不包括tcp的控制消息，比如建立连接，维持连接和关闭连接的 相 应控制报文。即在A上要监听某个tcp端口，要向系统的tcp/ip协议栈请求端口资源， 收到client的报文后，并不修改其源/目的ip:port，而是另外再去和B建立tcp连接。

\*client和A之间的tcp连接 并不影响 A和B之间的tcp连接。即使Client和A的连接中断 了，A和B之间也有可能仍在连接。

\*也就是说A上面要有运行某个普通的进程去监听要转发的端口，当此端口有客户端连接时， 再启用某本地端口和B建立一个tcp连接。

当有多个客户端和A的监听的端口连接时，A就要和B建立多个tcp连接

\*与端口转发相比，端口代理的开销较大，对A的系统本身的资源有一定影响

**Linux**上一般是在防火墙进程里启用端口转发（port-forward），系统本身不用监听要转发的 tcp端口，不处理tcp连接的事情，用netstat命令看不到监听端口，系统的防火墙也不 用再另外允许此端口入站，因为它只是过路包，不入站

**Windows**上一般是在系统里使用svchost.exe进行端口代理（portProxy），系统要监听 要代 理的tcp端口，处理tcp连接的相关事宜，用netstat命令可见监听的端口，系统的防火 墙要另外指明允许此端口入站

**http(s)/socks** **代理**（一般指浏览器/web服务器的代理），分2种方向:

**⑤正向代理：**

\*Client访问的目标不管是谁（只要符合走代理的规则），就统统把流量发给代理服务器，让 代理帮Client去访问目标服务器，

\*Client到代理之间的（http(s)/tcp/udp）流量是封装在代理协议层之上的，也就是说套了一 层代理协议的壳，

\*Client知道自己是把流量发给了代理服务器

\*代理服务器收到流量后，再解开这个壳，得到（http(s)/tcp/udp）流量，再根据此流量里的 相关信息（如域名）去找目标服务器的ip，最后代理再把流量发给目标服务器。

\*目标服务器响应的报文是发给代理服务器，

\*代理收到响应报文后再根据映射关系，把响应报文里的数据发给Client

**⑥反向代理：**

\*Client就正常访问目标服务器B，Client并不知道自己访问的是代理，

\*结果B是一个反向代理服务器（比如Nginx）它把收到的client发来的流量再进行某些修改 （修改Http的报头字段，当然也可不修改），再把数据发给后面的真实服务器

\*后面的真实服务器的响应报文是发给反向代理，

\*反向代理收到响应报文后再根据映射关系，修改某些字段，

\*最后把响应报文里的数据发给Client

关于http/https/socks的代理，代理服务器A不仅要处理tcp的相关事宜，还可能要处理高层协议的事情，比如http层的报文过滤，报头修改等，当然有时域名的解析也是由代理服务器去做的。

**小结**：

以上几种情况，凡是带有“代理”2个字的，都是要由代理服务器/进程（即A设备）去处理tcp连接相关的事情，client和A的tcp连接建立后，A再和目标B建立连接，然后将这2个连接做个映射关系，在这2个连接里传输载荷数据，当然，这个载荷数据也可能会被A修改某些字段。若某个连接断开了，A会尽快断开另一连接。

作者：Cof-Lee

2020-09-10