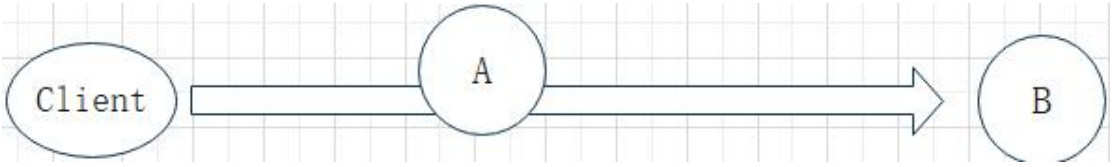
代理_端口转发_端口映射_NAT_概念释疑 2

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。

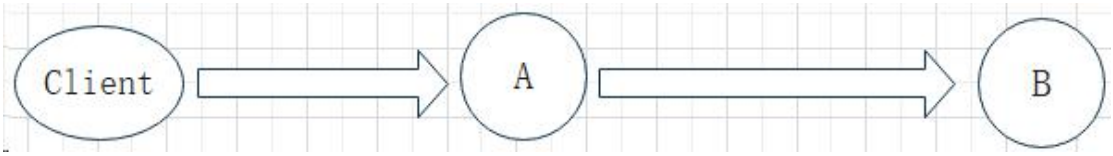
NAT 最初只是指 network address 的转换，即 ip 地址，后来也可以修改传输层的端口号，涉及到端口号的修改的操作也叫作 PAT（port address 转换），但我们仍习惯于把这二者的修改统称为 NAT。　NAT 是分 2 个方向的，对源 ip:port 的修改叫作 sNAT（源地址转换），对目标 ip:port 的修改叫作 dNAT（目的地址转换）

①sNAT （源 NAT）



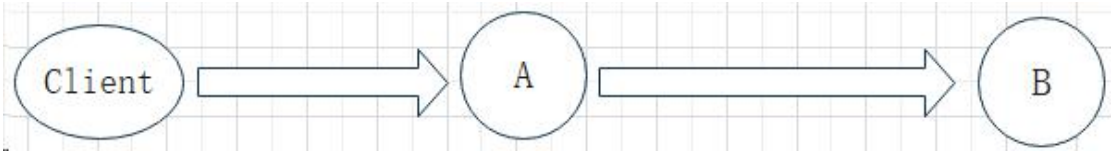
*sNAT 是指客户端发来的数据报文是给 B 的，但在路由上经过了 A 设备，
*然后 A 对这个流量的源 ip:port 做了修改，改成了 A 的 ip:port，并且 A 记录了这个转换的映射关系，
*B 收到数据后，响应的报文是发回给 A，因为从 B 那里看，数据的源 ip 是 A
*A 收到 B 发回的响应报文后，再根据 之前保存的映射关系，把响应报文的目的 ip:port 改为 Client
*最后响应报文走路由层 发到 Client

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：

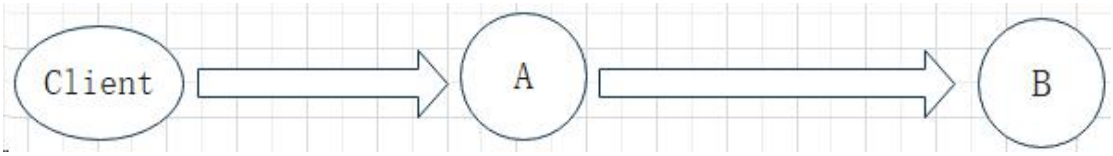
1.**只修改源 ip:port**，即 sNAT 源地址转换，其中 ip:port 可以只改 ip，也必须改 ip
响应报文一定原路返回

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