Engineer

NAT

🔀 the rewriting of IP addresses as packets pass a router

What it is

NAT, short for network address translation, is the rewriting of IP addresses in packets as they pass through a router, so that hosts numbered in one address space can talk to hosts numbered in another. RFC 2663, the 1999 document that fixed the vocabulary, describes it as mapping IP addresses from one realm to another, with the aim of transparent routing: the hosts on either side should not need to know that it happens.

The form almost everyone meets is what RFC 3022, published in January 2001, calls Traditional NAT. Its typical case is a small office or home with a single IP address from its provider, while the machines behind the router use private addresses from the RFC 1918 blocks 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16, which are not routed between organisations. The router rewrites the source address of each outgoing packet to its own public one and reverses the change on the replies. Basic NAT translates addresses alone, so only as many inside hosts can be out at once as there are public addresses in its pool. NAPT, network address port translation, also rewrites the TCP or UDP port or the ICMP identifier, which lets many inside hosts share one public address. RFC 3022 notes that Linux offered NAT under the name IP masquerade, and its firewall tools still use the word.

Changing an address breaks everything computed from it, so the router has to repair that as well. RFC 3022 requires the IP header checksum to be adjusted, and the TCP and UDP checksums too, because they cover a pseudo-header that contains the source and destination addresses. To send each reply to the right inside host, the router keeps a table of mappings: inside address and port on one side, outside address and port on the other. That table is state held in the middle of the network, and most of what goes wrong with NAT comes from it.

Traditional NAT works in one direction. RFC 3022 describes its sessions as outbound from the private network, with traffic the other way allowed only through static address maps for hosts chosen in advance. The idea now also runs one level up: RFC 6598, from April 2012, reserved 100.64.0.0/10 for carrier-grade NAT, numbering the links between a provider's NAT and its customers' equipment, which may translate once more.

Use the concept in a decision

Put "What is NAT?" to work in a real decision. We can help relate the concept to the system you are building or buying. Wi-Fi, firewalls and remote access set up so staff can work from anywhere and only the right people get in.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Managed 30 Ubuntu Linux VPS instances — 30 Linux servers managed, with recovery planned for

Managed 30 Ubuntu Linux VPS instances Get in touch →

Why it matters

RFC 3022 is candid about the trade. NAT can be installed without changing hosts or routers, but it takes away, in the RFC's words, "the end-to-end significance of an IP address", and makes up for it with more state in the network. Nearly every practical consequence of NAT follows from that sentence.

For a server, the source address of a connection is often not one machine but a router with a crowd behind it. A block or a rate limit aimed at one visitor lands on everyone who shares the address, and an abuse report names the router, not the culprit. RFC 3022 makes the same point from the inside: hiding private addresses, which it counts as a privacy benefit, also makes debugging harder, because a host that attacks other machines or sends spam has its address hidden in the NAT router.

State also means timers. A mapping lives only as long as the router remembers it. RFC 4787, a Best Current Practice from January 2007, says a UDP mapping must remain valid for at least two minutes and recommends five minutes or more. RFC 5382, its TCP counterpart from October 2008, sets 2 hours and 4 minutes for an idle established connection: keepalives go out every two hours by default, and the four minutes cover packets still in flight. These are floors for devices that follow the documents, and not every device does. A connection that stays quiet for longer than the device's own timer loses its mapping, and the next packet from outside has nowhere to go.

Two further constraints follow. RFC 3022 makes it mandatory that every packet of a session pass through the same NAT router, and where a network has more than one exit, it calls it essential for the NAT devices to share an identical table of translations. And protocols that carry addresses inside their payload need help from the NAT device, which works only while it can read that payload.

What it looks like in practice

On a Linux router, nftables performs NAT in chains of the type `nat`. The ruleset below is this article's illustration, built on the NAT examples in the nft(8) manual. Here 198.51.100.0/24 stands in for the inside network and 203.0.113.5 for the router's public address; both come from the documentation ranges, and a real inside network would normally use one of the RFC 1918 blocks.

```
add table nat
add chain nat prerouting { type nat hook prerouting priority dstnat; }
add chain nat postrouting { type nat hook postrouting priority srcnat; }
add rule nat postrouting oif eth0 ip saddr 198.51.100.0/24 snat to 203.0.113.5
add rule nat prerouting iif eth0 tcp dport 443 dnat to 198.51.100.20
```

The first rule is the everyday case: packets from the inside network that leave through eth0 get 203.0.113.5 as their source. For a dynamic public address, the manual offers `masquerade`, which uses whatever address the outgoing interface has at the moment. The second rule is a static map the other way: TCP connections that arrive on eth0 for port 443 go to 198.51.100.20 inside. So few rules are enough because only the first packet of a connection passes through a nat chain; its decision is stored in the connection tracking entry, and every later packet is rewritten the same way.

The other side of the picture is a host behind a NAT that it does not control, and WireGuard has a setting for exactly that. In this second illustration of mine, an inside host keeps a tunnel open to a peer on the outside:

```
[Peer]
PublicKey = <the outside peer's public key>
Endpoint = 192.0.2.44:51820
AllowedIPs = 192.0.2.128/25
PersistentKeepalive = 25
```

The wg(8) manual describes `PersistentKeepalive` as an interval of 1 to 65535 seconds for sending an authenticated empty packet, so that a stateful firewall or NAT mapping stays valid. It suggests 25 seconds for an interface behind NAT that rarely sends anything itself and may receive traffic at any time, and leaves the setting off by default. The outside peer needs nothing special: its `Endpoint` for this host is updated to the most recent source address and port of correctly authenticated packets, which is exactly what the NAT assigned.

Tunnels like these are where NAT trouble tends to surface. The portfolio describes eight WireGuard and IPsec connections managed and troubleshot from 2022 to 2024 across Google Cloud and Linux systems. That page says the tunnels were fragile and awkward to diagnose when they dropped, and that recurring routing and key-exchange problems were fixed at the root instead of with a restart, after which the interruptions stopped. It does not say whether NAT played a part, so the two examples above belong to this article, not to that project.

Where it goes wrong

Quiet connections that stop answering. A tunnel, a database connection or a push channel that sends nothing for a while loses its mapping and breaks without either end having closed it. The symptom looks random, because it depends on how long the line was idle. Send keepalives from the inside more often than the shortest timer on the path expires, as `PersistentKeepalive` does, and on a NAT you run, keep the timers at or above the RFC 4787 and RFC 5382 minimums. RFC 4787 requires outbound packets to refresh a mapping and warns that inbound refresh lets an attacker keep one alive forever, so the inside end should do the refreshing.

Expecting connections to come in. Traditional NAT is outbound by design, so a service on an inside host is unreachable until someone writes a static map for it. Add a `dnat` rule for exactly the ports the service needs, or turn the direction around and let the inside host connect out, as the keepalive above does.

Missing hairpinning. Two inside hosts that reach each other through the outside address of one of them need the NAT to turn the packet around. RFC 4787 requires NAT devices to support this hairpinning and to present the turned-around packet with the external source address and port. Test that path deliberately, and where a device fails it, have inside hosts use the inside address.

Addresses inside the payload. FTP is the classic case: RFC 2663 notes that its `PORT` command and `PASV` response carry an IP address and port as text, which a NAT that only rewrites headers leaves wrong. NAT devices patch such protocols with an Application Level Gateway (ALG), and RFC 3022 calls the one for FTP the most popular. An ALG must read the payload, so encryption disables it unless the gateway holds the key, and RFC 3022 adds that end-to-end IPsec cannot be assumed across a NAT. Prefer protocols that keep addresses out of the payload; where an old one must cross a NAT you run, the nft(8) manual shows a connection tracking helper for FTP.

Logs that only name the router. Behind NAT, a server's access log records the shared address, which on its own cannot single out one inside host. Log the source port and an exact timestamp next to the address, and keep clocks synchronised, so that a complaint about one connection can be matched against the translation on the router.

Fragments under NAPT. Only the first fragment of a packet carries the TCP or UDP header, so RFC 3022 warns that translating outgoing fragments with NAPT is bound to fail, and that two inside hosts sending fragments with the same identifier to the same destination can corrupt both sessions. Keep packets small enough to cross the path whole.

Treating NAT as a firewall. The one-way nature of NAT is easy to mistake for protection. RFC 4787 acknowledges that NATs are often deployed for security goals, but says plainly that the way a NAT maps makes no difference to its security, which is fully determined by its filtering behaviour: which packets it lets in. Write explicit filter rules for what may enter, and pair every `dnat` rule with a rule that allows only the traffic it was meant for.

Further reading

- RFC 3022: Traditional IP Network Address Translator (Traditional NAT) — the 2001 description of Basic NAT and NAPT, with an honest list of their costs.
- RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations — the vocabulary of NAT, from address realms to transparent routing and ALGs.
- RFC 4787: Network Address Translation (NAT) Behavioral Requirements for Unicast UDP — how a NAT should map, time out and hairpin UDP traffic.
- RFC 5382: NAT Behavioral Requirements for TCP — the TCP counterpart, including the idle timeouts for established and closing connections.

- wg(8): set and retrieve configuration of WireGuard interfaces — the WireGuard manual, with `PersistentKeepalive` and the advice for hosts behind NAT.
- nft(8): nftables manual — the reference for nftables, including nat chains and the `snat`, `dnat` and `masquerade` statements.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is NAT?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/nat/