Engineer

CIDR

⊞ a way to write and route address blocks of any size

What it is

CIDR, short for Classless Inter-Domain Routing, is the scheme that describes a block of IP addresses by a starting address and the number of leading bits that every address in the block shares. Written down, it is the familiar slash: 198.51.100.0/24 means every address whose first 24 bits match 198.51.100.0, which is 256 addresses from 198.51.100.0 to 198.51.100.255. The number after the slash is called the prefix length, and the block itself is called a prefix.

The word classless points at what came before. The original IPv4 plan from RFC 791 had three sizes of network, and RFC 4632, the current description of CIDR, explains why that failed: Class C, with 254 usable host addresses, was too small for a mid-sized organisation, and Class B, with up to 65,534, far too large, yet Class B was the best fit available. As the network was commercialised in the late 1980s, demand for Class B grew at what looked like an exponential rate.

In January 1992 the IETF's ROAD group (Routing and Addressing) named three problems: Class B space running out, routing tables outgrowing what routers and their operators could manage, and the eventual exhaustion of IPv4 itself. The first two were expected to turn critical between 1993 and 1995. CIDR was the answer to those two, first proposed in RFC 1338 in June 1992 and then in RFC 1519 in September 1993, which RFC 4632 replaced in August 2006. It was never meant to solve the third, only to buy time.

In CIDR notation, RFC 4632 writes a prefix as the usual four-part address, a slash, and a number from 0 to 32 giving the significant bits. The older netmask says the same with those bits set to one and the rest to zero, so /24 is 255.255.255.0. The one constraint the RFC keeps is that the mask must be contiguous, and within it a prefix can describe any power-of-two block. The RFC's table runs from /32, a host

route of one address, through /31 for a point-to-point link with two, /24 with 256 and /16 with 65,536, down to 0.0.0.0/0, all 4,294,967,296 addresses, used as the default route.

The notation outlived the address family it was built for: RFC 4291, which sets out how IPv6 addresses are structured, writes IPv6 prefixes the same way, so 2001:db8::/32 reads exactly like its IPv4 cousins.

Use the concept in a decision

Put “What is CIDR?” 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

CIDR solved its original problems by aggregation. In RFC 4632’s account, summarising routes is the only widely understood way for a packet-switched network to carry less routing state. A provider that holds one large block and numbers its customers from small pieces of it can announce a single route for the whole block, and the rest of the Internet needs nothing more to reach every customer inside it. RFC 4632 adds that address assignment itself had to change to make that possible, and that the part of the old Class C space which cannot be aggregated, known as the swamp, still held an estimated 15,000 routing entries in 2006.

Aggregation works because of a single rule that every router follows. In RFC 4632’s words: “Forwarding in the Internet is done on a longest-match basis.” When more than one prefix covers a destination, the most specific one wins. That is how an organisation that changes provider without renumbering stays reachable: the new provider announces the organisation’s own small prefix, which the RFC describes as punching a hole in the old provider’s aggregate, and it is preferred over the aggregate because it is the longer match. Planned to last three to five years, CIDR had, by the RFC’s account, been thoroughly deployed by 2005.

Few engineers configure inter-domain routing, but almost all of them write prefixes: in a firewall rule that admits an office network, in a route table on a Linux server, and in WireGuard, where each peer's AllowedIPs list of prefixes decides both which source addresses are accepted from that peer and where traffic for it is sent. In each place, one wrong digit after the slash silently admits or excludes thousands of hosts.

What it looks like in practice

Python's standard `ipaddress` module understands the notation and is a quick way to check arithmetic before it goes into a rule. The session below is this article's illustration, run on Python 3.15 with an address from the 198.51.100.0/24 documentation range:

```
>>> import ipaddress
>>> net = ipaddress.ip_network("198.51.100.0/24")
>>> net.netmask, net.num_addresses
(IPv4Address('255.255.255.0'), 256)
>>> for sub in net.subnets(new_prefix=26):
...     print(sub)
...
198.51.100.0/26
198.51.100.64/26
198.51.100.128/26
198.51.100.192/26
>>> ipaddress.ip_network("198.51.100.70/26")
Traceback (most recent call last):
  ...
ValueError: 198.51.100.70/26 has host bits set
>>> ipaddress.ip_network("198.51.100.70/26", strict=False)
IPv4Network('198.51.100.64/26')
>>> half = ipaddress.ip_network("198.51.100.128/25")
>>> ipaddress.ip_address("198.51.100.200") in half
True
>>> half.subnet_of(net)
True
```

A /24 splits into four /26 blocks of 64 addresses, each starting on a multiple of 64. Asking for 198.51.100.70/26 fails, because the block containing .70 starts at .64 and the bits after the 26th are not zero; with `strict=False` the module zeroes them and returns the block .70 belongs to. The last two lines are the questions a reviewer should ask of any pair of prefixes: does this address fall inside, and is one block wholly inside the other.

That second question matters most in a packet filter, which does not use longest match. It reads its rules in order and stops at the first that decides. The nft(8) manual says that `accept` and `drop` terminate evaluation of the chain, and that `add` appends a rule to the end of a chain while `insert` puts it at the beginning. In this second illustration of mine, the aim is to let a /24 reach SSH while keeping its upper half out:

```
add table inet filter
add chain inet filter input { type filter hook input priority filter; }
add rule inet filter input ip saddr 198.51.100.0/24 tcp dport 22 accept
add rule inet filter input ip saddr 198.51.100.128/25 tcp dport 22 drop
```

Both rules look right, and a listing shows both. But every address in the /25 is also in the /24, so the first rule accepts those packets and the drop below it is never reached. In routing, the more specific prefix would win wherever it sat; in a filter, it wins only if it comes first. Swapping the two lines, or adding the narrower rule with `insert`, gives the intended result. `nft -a list ruleset` prints every rule with its handle, and both `add` and `insert` accept a handle to place a new rule next to an existing one.

The portfolio entry Verified firewall rules by position, from August 2026, describes the same trap from the operating side. A host ran for a year with a rate limit that no packet ever reached, because it sat below a broader rule that matched first, and every tool that printed a summary showed the limit correctly. The checks now read the numbered rule list and the live filter chain and assert their order. That page does not say how the broader rule was written or whether address prefixes were involved, so the lines above illustrate the principle, not that host's rules.

Where it goes wrong

Host bits in the address. 198.51.100.70/26 is an address with a mask, not a block. Python rejects it by default, but with `strict=False` rounds it down to 198.51.100.64/26, which may not be the hosts the author meant. Write the network address of the block, and parse strictly so the mistake shows up as an error.

A bare address where a block was meant. Python's documentation says a network given without a mask is taken as /32, so an office's address typed without its prefix admits one host. Always write the prefix length, even /32, so a reader can tell a host route from a forgotten slash.

A broad prefix above a narrow one. As the illustration shows, a first-match filter lets the earlier rule shadow any later rule whose prefix it contains. Put the most specific rules first, and when rules are generated, test each pair with a containment check such as `subnet_of` or `overlaps` and fail the build if a later rule is unreachable.

A prefix length that is a few bits too short. Each bit removed doubles the block, so a /16 typed for a /24 admits 256 times as many addresses, and /0 admits every address there is. RFC 4632 records that operator error and other events have caused /1 and /2 routes to be observed in some networks in the past. Treat any rule shorter than the block you own as a review item, and reserve 0.0.0.0/0 and ::/0 for rules that are meant to match everything.

Counting hosts as addresses. A /24 has 256 addresses, but on an ordinary network the first is the network address and the last the broadcast address, so Python's `hosts()` returns 254 of them. On point-to-point links that waste adds up: RFC 3021 notes that a /30 spends four addresses on two hosts, and lets such links use a /31 instead, which saves 1,000 addresses across 500 links. Size a block for the hosts it must hold plus those reserved addresses, and use /31 only on equipment that supports RFC 3021.

An aggregate with no discard route. A router that announces a summary block but has lost one of the smaller prefixes inside it can send traffic for the missing piece back along a less specific route, such as the default, and so create a forwarding loop. RFC 4632 requires the router that generates the aggregate to discard packets matching it but no more specific route, which in practice means installing a discard or null route for every aggregate it announces.

Allow-lists that cover only IPv4. In nftables, `ip saddr` reads the IPv4 header and `ip6 saddr` the IPv6 one, and in Python, collapsing a mixed list raises a `TypeError` because the two versions cannot be combined. A list of IPv4 prefixes says nothing about IPv6 clients, which then fall through to whatever comes next. Keep IPv4 and IPv6 prefixes side by side for every rule, and check both when you test.

Further reading

- RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan — the current description of CIDR, with its history, the prefix table and the rules for aggregation.
- RFC 3021: Using 31-Bit Prefixes on IPv4 Point-to-Point Links — why a /30 spends half its addresses on a point-to-point link, and how a /31 avoids it.
- RFC 4291: IP Version 6 Addressing Architecture — the IPv6 addressing rules, including prefixes written in the same slash notation.
- ipaddress: IPv4/IPv6 manipulation library — Python's reference for parsing, splitting, comparing and collapsing prefixes.
- nft(8): nftables manual — the reference for nftables, including how rules are ordered and which verdicts end a chain.
- wg(8): set and retrieve configuration of WireGuard interfaces — the WireGuard manual, where AllowedIPs is defined as a list of addresses with CIDR masks.

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 CIDR?" 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/cidr/