Engineer

CIDR

⊞ en måde at skrive og route adresseblokke af alle størrelser

Hvad det er

CIDR, en forkortelse af Classless Inter-Domain Routing, er den måde, man beskriver en blok af IP-adresser på: med en startadresse og det antal forreste bit, som alle adresser i blokken har til fælles. Skrevet ned er det den velkendte skråstreg. 198.51.100.0/24 betyder alle adresser, hvis første 24 bit er de samme som i 198.51.100.0, altså 256 adresser fra 198.51.100.0 til 198.51.100.255. Tallet efter skråstregen kaldes præfikslængden, og selve blokken kaldes et præfiks.

Ordet classless, klasseløs, peger på det, der kom før. Den oprindelige IPv4-plan fra RFC 791 havde tre størrelser netværk, og RFC 4632, den gældende beskrivelse af CIDR, forklarer, hvorfor det gik galt: Klasse C med højst 254 værtsadresser var for lille til en mellemstor organisation, og klasse B med op til 65.534 var alt for stor, men alligevel det bedste, der fandtes. Da nettet blev kommercialiseret i slutningen af 1980'erne, voksede efterspørgslen efter klasse B i noget, der lignede et eksponentielt tempo.

IETF's ROAD-gruppe (Routing and Addressing) mødtes i januar 1992 og pegede på tre problemer: at adresserummet i klasse B var ved at slippe op, at routingtabellerne voksede ud over, hvad routere og deres operatører kunne håndtere, og at IPv4 selv en dag ville løbe tør for adresser. De to første ventede man ville blive kritiske et sted mellem 1993 og 1995. CIDR var svaret på dem, først foreslået i RFC 1338 i juni 1992 og siden i RFC 1519 i september 1993, som RFC 4632 afløste i august 2006. Det tredje problem forsøgte CIDR aldrig at løse; det skulle kun købe tid.

I CIDR-notation skriver RFC 4632 et præfiks som den sædvanlige adresse i fire dele, en skråstreg og et tal fra 0 til 32, der angiver de betydende bit. Den ældre netmaske siger det samme med de bit sat til ét og resten til nul, så /24 er 255.255.255.0. Den eneste begrænsning, RFC'en holder fast i, er, at masken skal

være sammenhængende, og inden for den kan et præfiks beskrive enhver blok, hvis størrelse er en potens af to. RFC'ens tabel går fra /32, en værtsrute med én adresse, over /31 til en punkt-til-punkt-forbindelse med to, /24 med 256 og /16 med 65.536, ned til 0.0.0.0/0 med alle 4.294.967.296 adresser, som bruges som standardrute.

Notationen har overlevet den adressefamilie, den blev bygget til: RFC 4291, der fastlægger, hvordan IPv6-adresser er opbygget, skriver IPv6-præfikser på samme måde, så 2001:db8::/32 læses præcis som sine IPv4-fætre.

Brug begrebet i en beslutning

Brug »Hvad er CIDR?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Wi-Fi, firewalls og fjernadgang sat op, så medarbejdere kan arbejde hvor som helst, og kun de rette kommer ind.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har administreret 30 Ubuntu Linux VPS-instanser — 30 Linux-servere driftet, med en plan for gendannelse

Har administreret 30 Ubuntu Linux VPS-instanser Kontakt os →

Hvorfor det betyder noget

CIDR løste sine oprindelige problemer ved aggregering. RFC 4632 kalder aggregering af information den eneste almindeligt forståede metode til at begrænse routingtilstanden i et pakkekoblet netværk. En udbyder, der har én stor blok og nummererer sine kunder fra små stykker af den, kan annoncere en enkelt rute for hele blokken, og resten af internettet behøver ikke mere for at nå hver eneste kunde i den. RFC 4632 tilføjer, at selve tildelingen af adresser måtte lægges om for at gøre det muligt, og at den del af det gamle klasse C-rum, som ikke kan aggregeres, og som på engelsk kaldes the swamp, sumpen, i 2006 blev anslået til omkring 15.000 routingposter.

Aggregering virker på grund af en enkelt regel, som alle routere følger. Med RFC 4632's ord: »Videresendelse på internettet sker på grundlag af længste match.« Når flere præfikser dækker samme destination, vinder det mest specifikke. Sådan forbliver en organisation, der skifter udbyder uden at omnummerere, tilgængelig: Den nye udbyder annoncerer organisationens eget lille præfiks, hvad RFC'en beskriver som at slå hul i den gamle udbyders aggregat, og det foretrækkes frem for aggregatet, fordi det er det længere match. CIDR var tænkt til at holde i tre til fem år, men var ifølge RFC'en fuldt udrullet i 2005.

Få ingeniører sætter routing mellem domæner op, men næsten alle skriver præfikser: i en firewallregel, der lukker et kontornetværk ind, i en routingtabel på en Linux-server og i WireGuard, hvor hver modparts AllowedIPs-liste af præfikser afgør både, hvilke kildeadresser der accepteres fra den modpart, og hvor trafik til den sendes. Alle tre steder kan ét forkert ciffer efter skråstregen i stilhed lukke tusindvis af værter ind eller ude.

Sådan ser det ud i praksis

Pythons standardmodul `ipaddress` forstår notationen og er en hurtig måde at tjekke regnestykket på, før det ender i en regel. Sessionen herunder er denne artikels illustration, kørt på Python 3.15 med adresser fra dokumentationsområdet 198.51.100.0/24:

```
>>> 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
```

En /24 deler sig i fire /26-blokke med 64 adresser i hver, og hver af dem starter ved et multiplum af 64. Forsøget med 198.51.100.70/26 fejler, fordi blokken, der indeholder .70, starter ved .64, og bittene efter den 26. ikke er nul; med `strict=False` sætter modulet dem til nul og returnerer den blok, .70 hører til. De to sidste linjer er de spørgsmål, man bør stille til ethvert par af præfikser, når man gennemgår regler: Ligger adressen inden for, og ligger den ene blok helt inden i den anden?

Det andet spørgsmål betyder mest i et pakkefilter, for det bruger ikke længste match. Det læser sine regler i rækkefølge og standser ved den første, der afgør sagen. Manualen nft(8) siger, at `accept` og `drop` afslutter evalueringen af kæden, og at `add` føjer en regel til sidst i en kæde, mens `insert` sætter den forrest. I denne artikels anden illustration er målet at lade en /24 nå SSH, men holde dens øverste halvdel ude:

```
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
```

Begge regler ser rigtige ud, og en oversigt viser dem begge. Men hver adresse i /25-blokken ligger også i /24-blokken, så den første regel accepterer de pakker, og drop-reglen under den nås aldrig. I routing ville det mere specifikke præfiks vinde, uanset hvor det stod; i et filter vinder det kun, hvis det kommer først. Byt om på de to linjer, eller tilføj den smalle regel med `insert`, så får du det tilsigtede resultat. `nft -a list ruleset` udskriver hver regel med dens handle, og både `add` og `insert` tager imod et handle, så en ny regel kan placeres ved siden af en eksisterende.

Porteføljeposten Har verificeret firewallregler efter position fra august 2026 beskriver den samme fælde set fra driftssiden. En vært kørte i et år med en hastighedsbegrænsning, som ingen pakke nogensinde nåede, fordi den lå under en bredere regel, der matchede først, og hvert eneste værktøj, der var involveret, havde udskrevet begrænsningen korrekt i hele den tid. Tjekkene læser nu den nummererede regeloversigt og den aktive filterkæde og efterprøver rækkefølgen. Siden siger ikke, hvordan den bredere regel var skrevet, eller om adressepræfikser spillede ind, så linjerne ovenfor illustrerer princippet og ikke den værts regler.

Hvor det går galt

Værtsbits i adressen. 198.51.100.70/26 er en adresse med en maske, ikke en blok. Python afviser den som standard, men med `strict=False` runder modulet den ned til 198.51.100.64/26, som måske ikke er de værter, forfatteren mente. Skriv blokkens netværksadresse, og fortolk strengt, så tastefejlen viser sig som en fejlmelding.

En enkelt adresse, hvor en blok var ment. Pythons dokumentation siger, at et netværk uden maske tolkes som /32, så et kontors adresse skrevet uden præfiks lukker én vært ind. Skriv altid præfikslængden, også /32, så en læser kan skelne en bevidst værtsrute fra en glemt skråstreg.

Et bredt præfiks over et smalt. Som illustrationen viser, lader et filter med første match den tidligere regel overskygge enhver senere regel, hvis præfiks den indeholder. Sæt de mest specifikke regler først, og når reglerne genereres, så test hvert par med et indeholdelsestjek som `subnet_of` eller `overlaps`, og lad buildet fejle, hvis en senere regel ikke kan nås.

En præfikslængde, der er et par bit for kort. Hver bit, der fjernes, fordobler blokken, så en /16 skrevet i stedet for en /24 lukker 256 gange så mange adresser ind, og /0 lukker alle adresser ind. RFC 4632 noterer, at operatørfejl og andre begivenheder tidligere har fået /1- og /2-ruter til at dukke op i nogle netværk. Behandl enhver regel, der er kortere end den blok, du selv har, som et punkt til gennemgang, og hold 0.0.0.0/0 og ::/0 til regler, der skal matche alt.

Værter talt som adresser. En /24 har 256 adresser, men på et almindeligt netværk er den første netværksadressen og den sidste broadcastadressen, så Pythons `hosts()` returnerer 254 af dem. På punkt-til-punkt-forbindelser løber spildet op: RFC 3021 påpeger, at en /30 bruger fire adresser på to værter, og lader sådanne forbindelser bruge en /31 i stedet, hvilket sparer 1.000 adresser på 500 forbindelser. Dimensionér en blok efter de værter, den skal rumme, plus de reserverede adresser, og brug kun /31 på udstyr, der understøtter RFC 3021.

Et aggregat uden discard-rute. En router, der annoncerer en samleblok, men har mistet et af de mindre præfikser inden i den, kan sende trafik til det manglende stykke tilbage ad en mindre specifik rute, for eksempel standardruten, og dermed skabe en routingløkke. RFC 4632 kræver, at den router, der danner aggregatet, kasserer pakker, som matcher det, men ingen mere specifik rute, og i praksis betyder det en discard- eller null-rute for hvert aggregat, den annoncerer.

Tilladelseslister, der kun dækker IPv4. I nftables læser `ip saddr` IPv4-headeren og `ip6 saddr` IPv6-headeren, og i Python udløser det en `TypeError` at samle en blandet liste, fordi de to versioner ikke kan kombineres. En liste over IPv4-præfikser siger intet om IPv6-klienter, som så falder igennem til det, der kommer bagefter. Hold IPv4- og IPv6-præfikser side om side i hver regel, og tjek begge, når du tester.

Læs mere

- RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan — den gældende beskrivelse af CIDR med historien, præfikstabellen og reglerne for aggregering.
- RFC 3021: Using 31-Bit Prefixes on IPv4 Point-to-Point Links — hvorfor en /30 bruger halvdelen af sine adresser på en punkt-til-punkt-forbindelse, og hvordan en /31 undgår det.
- RFC 4291: IP Version 6 Addressing Architecture — reglerne for IPv6-adresser, herunder præfikser skrevet med den samme skråstregsnotation.
- ipaddress: IPv4/IPv6 manipulation library — Pythons dokumentation for at fortolke, opdele, sammenligne og samle præfikser.
- nft(8): nftables manual — opslagsværket for nftables, herunder hvordan regler ordnes, og hvilke afgørelser der afslutter en kæde.
- wg(8): set and retrieve configuration of WireGuard interfaces — WireGuard-manualen, hvor AllowedIPs er defineret som en liste af adresser med CIDR-masker.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er CIDR?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/cidr/