Engineer

CIDR

⊞ en måte å skrive og rute adresseblokker av alle størrelser

Hva det er

CIDR, en forkortelse for Classless Inter-Domain Routing, er måten man beskriver en blokk med IP-adresser på: med en startadresse og antallet innledende biter som alle adressene i blokken har felles. Skrevet ned er det den velkjente skråstreken. 198.51.100.0/24 betyr alle adresser der de første 24 bitene er de samme som i 198.51.100.0, altså 256 adresser fra 198.51.100.0 til 198.51.100.255. Tallet etter skråstreken kalles prefikslengden, og selve blokken kalles et prefiks.

Ordet classless, klasseløs, peker på det som kom før. Den opprinnelige IPv4-planen fra RFC 791 hadde tre størrelser nettverk, og RFC 4632, den gjeldende beskrivelsen av CIDR, forklarer hvorfor det ikke holdt: Klasse C, med høyst 254 vertsadresser, var for liten for en mellomstor organisasjon, og klasse B, med opptil 65 534, altfor stor, men likevel det beste som fantes. Da nettet ble kommersialisert på slutten av 1980-tallet, vokste etterspørselen etter klasse B i det som så ut som et eksponentielt tempo.

IETFs ROAD-gruppe (Routing and Addressing) møttes i januar 1992 og pekte på tre problemer: at adresserommet i klasse B holdt på å gå tomt, at rutingtabellene vokste ut over det rutere og de som driftet dem, kunne håndtere, og at IPv4 selv en dag ville gå tom for adresser. De to første ventet man ville bli kritiske en gang mellom 1993 og 1995. CIDR var svaret på dem, først foreslått i RFC 1338 i juni 1992 og deretter i RFC 1519 i september 1993, som RFC 4632 erstattet i august 2006. Det tredje problemet prøvde CIDR aldri å løse; det skulle bare kjøpe tid.

I CIDR-notasjon skriver RFC 4632 et prefiks som den vanlige adressen i fire deler, en skråstrek og et tall fra 0 til 32 som angir de signifikante bitene. Den eldre nettmasken sier det samme med disse bitene satt til én og resten til null, så /24

er 255.255.255.0. Den eneste begrensningen RFC-en holder fast ved, er at masken må være sammenhengende, og innenfor den kan et prefiks beskrive enhver blokk med en størrelse som er en potens av to. Tabellen i RFC-en går fra /32, en vertsrute med én adresse, via /31 for 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 brukes som standardrute.

Notasjonen har overlevd adressefamilien den ble laget for: RFC 4291, som fastsetter hvordan IPv6-adresser er bygd opp, skriver IPv6-prefikser på samme måte, så 2001:db8::/32 leses akkurat som sine IPv4-søskenbarn.

Bruk begrepet i en beslutning

Bruk «Hva er CIDR?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Wi-Fi, brannmurer og fjerntilgang satt opp slik at de ansatte kan jobbe hvor som helst, og bare de rette personene slipper inn.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Administrerte 30 Ubuntu Linux-VPS-instanser — 30 Linux-servere administrert, med gjenoppretting planlagt på forhånd

Administrerte 30 Ubuntu Linux-VPS-instanser Ta kontakt →

Hvorfor det er viktig

CIDR løste de opprinnelige problemene sine med aggregering. RFC 4632 kaller aggregering av informasjon den eneste allment forståtte metoden for å redusere rutingtilstanden i et pakkesvitsjet nettverk. En leverandør som har én stor blokk og nummererer kundene sine fra små biter av den, kan annonsere én enkelt rute for hele blokken, og resten av internett trenger ikke mer for å nå hver eneste kunde i den. RFC 4632 legger til at selve tildelingen av adresser måtte legges om for å gjøre det mulig, og at den delen av det gamle klasse C-rommet som ikke kan aggregeres, og som på engelsk kalles the swamp, sumpen, i 2006 ble anslått til rundt 15 000 rutingoppføringer.

Aggregering virker på grunn av én enkelt regel som alle rutere følger. Med ordene til RFC 4632: «Videresending på internett skjer etter prinsippet om lengste treff.» Når flere prefikser dekker samme mål, vinner det mest spesifikke. Slik forblir en organisasjon som bytter leverandør uten å nummerere om, tilgjengelig: Den nye leverandøren annonserer organisasjonens eget lille prefiks, noe RFC-en beskriver som å slå hull i den gamle leverandørens aggregat, og det foretrekkes framfor aggregatet fordi det er det lengre treffet. CIDR var tenkt å vare i tre til fem år, men var ifølge RFC-en grundig utrullet i 2005.

Få ingeniører setter opp ruting mellom domener, men nesten alle skriver prefikser: i en brannmurregel som slipper inn et kontornettverk, i en rutingtabell på en Linux-server og i WireGuard, der AllowedIPs-listen med prefikser for hver motpart avgjør både hvilke kildeadresser som godtas fra den motparten, og hvor trafikk til den sendes. Alle tre steder kan ett feil siffer etter skråstreken i det stille slippe inn eller stenge ute tusenvis av verter.

Slik ser det ut i praksis

Pythons standardmodul `ipaddress` forstår notasjonen og er en rask måte å sjekke regnestykket på før det havner i en regel. Økten nedenfor er denne artikkelens illustrasjon, kjørt på Python 3.15 med adresser fra dokumentasjonsområdet 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 seg i fire /26-blokker med 64 adresser i hver, og hver av dem starter på et multiplum av 64. Forsøket med 198.51.100.70/26 feiler fordi blokken som inneholder .70, starter på .64, og bitene etter den 26. ikke er null; med `strict=False` setter modulen dem til null og returnerer blokken .70 hører til. De to siste linjene er spørsmålene man bør stille til ethvert par av prefikser når man går gjennom regler: Ligger adressen innenfor, og ligger den ene blokken helt inne i den andre?

Det andre spørsmålet betyr mest i et pakkefilter, for det bruker ikke lengste treff. Det leser reglene sine i rekkefølge og stopper ved den første som avgjør saken. Manualen nft(8) sier at `accept` og `drop` avslutter evalueringen av kjeden, og at `add` legger en regel til sist i en kjede, mens `insert` setter den først. I denne artikkelens andre illustrasjon er målet å la en /24 nå SSH, men holde den øvre halvdelen ute:

```
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 reglene ser riktige ut, og en oversikt viser begge. Men hver adresse i /25-blokken ligger også i /24-blokken, så den første regelen godtar trafikken derfra, og drop-regelen under den nås aldri. I ruting ville det mer spesifikke prefikset vinne uansett hvor det sto; i et filter vinner det bare hvis det kommer først. Bytt om de to linjene, eller legg til den smale regelen med `insert`, så får du det tiltenkte resultatet. `nft -a list ruleset` skriver ut hver regel med sitt handle, og både `add` og `insert` tar imot et handle, slik at en ny regel kan plasseres ved siden av en eksisterende.

Porteføljeoppføringen Verifiserte brannmurregler etter plassering fra august 2026 beskriver den samme fella sett fra driftssiden. En vert gikk i et år med en hastighetsbegrensning som ingen pakke noen gang nådde, fordi den lå under en bredere regel som slo til først, og alle verktøyene som var involvert, hadde skrevet ut begrensningen korrekt hele tiden. Sjekkene leser nå den nummererte regeloversikten og den aktive filterkjeden og kontrollerer rekkefølgen. Siden sier ikke hvordan den bredere regelen var skrevet, eller om adresseprefikser spilte inn, så linjene ovenfor illustrerer prinsippet og ikke reglene på den verten.

Hvor det går galt

Vertsbiter i adressen. 198.51.100.70/26 er en adresse med en maske, ikke en blokk. Python avviser den som standard, men med `strict=False` runder modulen den ned til 198.51.100.64/26, som kanskje ikke er de vertene forfatteren mente. Skriv nettverksadressen til blokken, og tolk strengt, slik at slurvefeilen viser seg som en feilmelding.

En enkelt adresse der en blokk var ment. Dokumentasjonen til Python sier at et nettverk uten maske tolkes som /32, så en kontoradresse skrevet uten prefiks slipper inn én vert. Skriv alltid prefikslengden, også /32, slik at en leser kan skille en bevisst vertsrute fra en glemt skråstrek.

Et bredt prefiks over et smalt. Som illustrasjonen viser, lar et filter med første treff den tidligere regelen skygge for enhver senere regel med et prefiks den inneholder. Sett de mest spesifikke reglene først, og når reglene genereres, test hvert par med en sjekk av om det ene prefikset inneholder det andre, med `subnet_of` eller `overlaps`, og la bygget feile hvis en senere regel ikke kan nås.

En prefikslengde som er noen biter for kort. Hver bit som fjernes, dobler blokken, så en /16 skrevet i stedet for en /24 slipper inn 256 ganger så mange adresser, og /0 slipper inn alle adresser som finnes. RFC 4632 noterer at operatørfeil og andre hendelser tidligere har fått /1- og /2-ruter til å dukke opp i enkelte nettverk. Behandle enhver regel som er kortere enn blokken du selv har, som et punkt til gjennomgang, og reserver 0.0.0.0/0 og ::/0 for regler som skal treffe alt.

Verter telt som adresser. En /24 har 256 adresser, men på et vanlig nettverk er den første nettverksadressen og den siste kringkastingsadressen, så `hosts()` i Python returnerer 254 av dem. På punkt-til-punkt-forbindelser hoper sløsingen seg opp: RFC 3021 påpeker at en /30 bruker fire adresser på to verter, og lar slike forbindelser bruke en /31 i stedet, noe som sparer 1 000 adresser på 500 forbindelser. Dimensjoner en blokk etter vertene den skal romme, pluss de reserverte adressene, og bruk /31 bare på utstyr som støtter RFC 3021.

Et aggregat uten discard-rute. En ruter som annonserer en samleblokk, men har mistet et av de mindre prefiksene inne i den, kan sende trafikk for den manglende biten tilbake langs en mindre spesifikk rute, for eksempel standardruten, og dermed lage en rutingsløyfe. RFC 4632 krever at ruteren som lager aggregatet, forkaster pakker som treffer det, men ingen mer spesifikk rute, og i praksis betyr det en discard- eller null-rute for hvert aggregat den annonserer.

Tillatelseslister som bare dekker IPv4. I nftables leser `ip saddr` IPv4-headeren og `ip6 saddr` IPv6-headeren, og i Python gir det en `TypeError` å slå sammen en blandet liste, fordi de to versjonene ikke kan kombineres. En liste med IPv4-

prefikser sier ingenting om IPv6-klienter, som da faller gjennom til det som kommer etterpå. Hold IPv4- og IPv6-prefikser side om side i hver regel, og sjekk begge når du tester.

Les mer

- RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan — den gjeldende beskrivelsen av CIDR, med historien, prefikstabellen og reglene for aggregering.
- RFC 3021: Using 31-Bit Prefixes on IPv4 Point-to-Point Links — hvorfor en /30 bruker halvparten av adressene sine på en punkt-til-punkt-forbindelse, og hvordan en /31 unngår det.
- RFC 4291: IP Version 6 Addressing Architecture — reglene for IPv6-adresser, også prefikser skrevet med den samme skråstreknotasjonen.
- ipaddress: IPv4/IPv6 manipulation library — dokumentasjonen til Python for å tolke, dele opp, sammenligne og slå sammen prefikser.
- nft(8): nftables manual — oppslagsverket for nftables, blant annet om hvordan regler ordnes, og hvilke avgjørelser som avslutter en kjede.
- wg(8): set and retrieve configuration of WireGuard interfaces — WireGuard-manualen, der AllowedIPs er definert som en liste med adresser med CIDR-masker.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er CIDR?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/cidr/