Engineer

CIDR

⊞ ett sätt att skriva och routa adressblock av valfri storlek

Vad det är

CIDR, en förkortning av Classless Inter-Domain Routing, är sättet att beskriva ett block av IP-adresser med en startadress och det antal inledande bitar som alla adresser i blocket har gemensamt. Nedskrivet är det det välbekanta snedstrecket. 198.51.100.0/24 betyder alla adresser vars första 24 bitar är desamma som i 198.51.100.0, alltså 256 adresser från 198.51.100.0 till 198.51.100.255. Talet efter snedstrecket kallas prefixlängden, och själva blocket kallas ett prefix.

Ordet classless, klasslös, pekar på det som fanns före. Den ursprungliga IPv4-planen från RFC 791 hade tre storlekar av nätverk, och RFC 4632, den gällande beskrivningen av CIDR, förklarar varför det inte höll: Klass C, med högst 254 värdadresser, var för litet för en medelstor organisation, och klass B, med upp till 65 534, alldeles för stort, men ändå det bästa som fanns. När nätet kommersialiserades i slutet av 1980-talet växte efterfrågan på klass B i vad som såg ut som exponentiell takt.

IETF:s ROAD-grupp (Routing and Addressing) möttes i januari 1992 och pekade ut tre problem: att adressrymden i klass B höll på att ta slut, att routningstabellerna växte bortom vad routrar och de som drev dem kunde hantera, och att IPv4 självt en dag skulle få slut på adresser. De två första väntades bli kritiska någon gång mellan 1993 och 1995. CIDR var svaret på dem, först föreslaget i RFC 1338 i juni 1992 och sedan i RFC 1519 i september 1993, som RFC 4632 ersatte i augusti 2006. Det tredje problemet försökte CIDR aldrig lösa; det skulle bara köpa tid.

I CIDR-notation skriver RFC 4632 ett prefix som den vanliga adressen i fyra delar, ett snedstreck och ett tal från 0 till 32 som anger de signifikanta bitarna. Den äldre nätmasken säger samma sak med de bitarna satta till ett och resten till noll,

så /24 är 255.255.255.0. Den enda begränsning RFC:n håller fast vid är att masken måste vara sammanhängande, och inom den kan ett prefix beskriva varje block vars storlek är en potens av två. Tabellen i RFC:n går från /32, en värdrutt med en adress, via /31 för en punkt-till-punkt-länk med två, /24 med 256 och /16 med 65 536, ned till 0.0.0.0/0 med alla 4 294 967 296 adresser, som används som standardrutt.

Notationen har överlevt adressfamiljen den byggdes för: RFC 4291, som fastställer hur IPv6-adresser är uppbyggda, skriver IPv6-prefix på samma sätt, så 2001:db8::/32 läses precis som sina IPv4-kusiner.

Använd begreppet i ett beslut

Använd ”Vad är CIDR?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Wi-Fi, brandväggar och fjärråtkomst uppsatta så att personalen kan arbeta varifrån som helst och bara rätt personer kommer in.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Förvaltade 30 VPS-instanser med Ubuntu Linux — 30 förvaltade Linux-servrar, med en plan för återställning

Förvaltade 30 VPS-instanser med Ubuntu Linux Hör av er →

Varför det spelar roll

CIDR löste sina ursprungliga problem genom aggregering. RFC 4632 kallar aggregering av information den enda allmänt förstådda metoden för att minska routningstillståndet i ett paketförmedlande nätverk. En leverantör som har ett stort block och numrerar sina kunder ur små bitar av det kan annonsera en enda rutt för hela blocket, och resten av internet behöver inget mer för att nå varje kund i det. RFC 4632 tillägger att själva tilldelningen av adresser måste läggas om för att göra det möjligt, och att den del av den gamla klass C-rymden som inte går att aggregera, på engelska kallad the swamp, träsket, år 2006 uppskattades till omkring 15 000 poster i routningstabellen.

Aggregering fungerar tack vare en enda regel som alla routrar följer. Med RFC 4632:s ord: ”Vidarebefordran på internet sker efter principen om längsta matchning.” När flera prefix täcker samma destination vinner det mest specifika. Så förblir en organisation som byter leverantör utan att numrera om nåbar: Den nya leverantören annonserar organisationens eget lilla prefix, vilket RFC:n beskriver som att slå hål i den gamla leverantörens aggregat, och det föredras framför aggregatet eftersom det är den längre matchningen. CIDR var tänkt att räcka i tre till fem år men var enligt RFC:n grundligt infört 2005.

Få ingenjörer sätter upp routning mellan domäner, men nästan alla skriver prefix: i en brandväggsregel som släpper in ett kontorsnät, i en routningstabell på en Linux-server och i WireGuard, där AllowedIPs-listan med prefix för varje motpart avgör både vilka källadresser som godtas från den motparten och vart trafik till den skickas. På alla tre ställena kan en enda felaktig siffra efter snedstrecket i tysthet släppa in eller stänga ute tusentals värdar.

Så ser det ut i praktiken

Pythons standardmodul `ipaddress` förstår notationen och är ett snabbt sätt att kontrollera räkningen innan den hamnar i en regel. Sessionen nedan är den här artikelns illustration, körd på Python 3.15 med adresser från dokumentationsblocket 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
```

Ett /24 delar sig i fyra /26-block med 64 adresser i varje, och vart och ett börjar på en multipel av 64. Försöket med 198.51.100.70/26 misslyckas eftersom blocket som innehåller .70 börjar på .64 och bitarna efter den 26:e inte är noll; med `strict=False` nollställer modulen dem och returnerar det block som .70 hör till. De två sista raderna är de frågor man bör ställa om varje par av prefix när man granskar regler: Ligger adressen inom blocket, och ligger det ena blocket helt inuti det andra?

Den andra frågan betyder mest i ett paketfilter, eftersom det inte använder längsta matchning. Det läser sina regler i ordning och stannar vid den första som avgör saken. Manualen nft(8) säger att `accept` och `drop` avslutar utvärderingen av kedjan, och att `add` lägger en regel sist i en kedja medan `insert` lägger den först. I den här artikelns andra illustration är målet att låta ett /24 nå SSH men hålla dess övre halva 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
```

Båda reglerna ser riktiga ut, och en listning visar båda. Men varje adress i /25-blocket ligger också i /24-blocket, så den första regeln godtar trafiken därifrån, och drop-regeln under den nås aldrig. I routning skulle det mer specifika prefixet vinna oavsett var det stod; i ett filter vinner det bara om det kommer först. Byt plats på de två raderna, eller lägg till den smala regeln med `insert`, så får du det avsedda resultatet. `nft -a list ruleset` skriver ut varje regel med dess handle, och både `add` och `insert` tar emot ett handle, så att en ny regel kan placeras intill en befintlig.

Portföljposten Verifierade brandväggsregler efter position från augusti 2026 beskriver samma fälla från driftsidan. En värd gick i ett år med en hastighetsbegränsning som inget paket någonsin nådde fram till, eftersom den låg under en bredare regel som matchade först, och varje inblandat verktyg hade skrivit ut begränsningen korrekt hela tiden. Kontrollerna läser nu den numrerade regellistan och den aktiva filterkedjan och verifierar ordningen. Sidan säger inte hur den bredare regeln var skriven eller om adressprefix spelade in, så raderna ovan illustrerar principen och inte den värdens regler.

Var det går fel

Värdbitar i adressen. 198.51.100.70/26 är en adress med en mask, inte ett block. Python avvisar den som standard, men med `strict=False` avrundar modulen den nedåt till 198.51.100.64/26, vilket kanske inte är de värdar som författaren menade. Skriv blockets nätverksadress, och tolka strikt så att slarvfelet syns som ett felmeddelande.

En ensam adress där ett block var tänkt. Pythons dokumentation säger att ett nätverk utan mask tolkas som /32, så en kontorsadress skriven utan prefix släpper in en enda värd. Skriv alltid prefixlängden, även /32, så att en läsare kan skilja en avsiktlig värdrutt från ett glömt snedstreck.

Ett brett prefix ovanför ett smalt. Som illustrationen visar låter ett filter med första träff den tidigare regeln skugga varje senare regel vars prefix den innehåller. Lägg de mest specifika reglerna först, och när reglerna genereras, testa varje par med en kontroll av om det ena prefixet innehåller det andra, med `subnet_of` eller `overlaps`, och låt bygget misslyckas om en senare regel inte kan nås.

En prefixlängd som är några bitar för kort. Varje bit som tas bort fördubblar blocket, så ett /16 skrivet i stället för ett /24 släpper in 256 gånger så många adresser, och /0 släpper in alla adresser som finns. RFC 4632 noterar att operatörsfel och andra händelser tidigare har gjort att /1- och /2-rutter dykt upp i vissa nätverk. Behandla varje regel som är kortare än blocket du själv har som en granskningspunkt, och reservera 0.0.0.0/0 och ::/0 för regler som ska matcha allt.

Värdar räknade som adresser. Ett /24 har 256 adresser, men på ett vanligt nätverk är den första nätverksadressen och den sista broadcastadressen, så `hosts()` i Python returnerar 254 av dem. På punkt-till-punkt-länkar blir spillet stort: RFC 3021 påpekar att ett /30 lägger fyra adresser på två värdar, och låter sådana länkar använda ett /31 i stället, vilket sparar 1 000 adresser på 500 länkar. Dimensionera ett block efter de värdar det ska rymma plus de reserverade adresserna, och använd /31 bara på utrustning som stöder RFC 3021.

Ett aggregat utan discard-rutt. En router som annonserar ett sammanfattande block men har tappat ett av de mindre prefixen inuti det kan skicka trafik för den saknade biten tillbaka längs en mindre specifik rutt, till exempel standardrutten, och på så vis skapa en routningsloop. RFC 4632 kräver att den router som skapar aggregatet kastar paket som matchar det men ingen mer specifik rutt, och i praktiken betyder det en discard- eller null-rutt för varje aggregat den annonserar.

Tillåtelselistor som bara täcker IPv4. I nftables läser `ip saddr` IPv4-huvudet och `ip6 saddr` IPv6-huvudet, och i Python ger det ett `TypeError` att slå ihop en blandad lista, eftersom de två versionerna inte kan kombineras. En lista med

IPv4-prefix säger ingenting om IPv6-klienter, som då faller igenom till det som kommer efteråt. Håll IPv4- och IPv6-prefix sida vid sida i varje regel, och kontrollera båda när du testar.

Läs mer

- RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan — den gällande beskrivningen av CIDR, med historiken, prefixtabellen och reglerna för aggregering.
- RFC 3021: Using 31-Bit Prefixes on IPv4 Point-to-Point Links — varför ett /30 lägger hälften av sina adresser på en punkt-till-punkt-länk och hur ett /31 undviker det.
- RFC 4291: IP Version 6 Addressing Architecture — reglerna för IPv6-adresser, också prefix skrivna med samma snedstrecksnotation.
- ipaddress: IPv4/IPv6 manipulation library — Pythons dokumentation för att tolka, dela upp, jämföra och slå ihop prefix.
- nft(8): nftables manual — referensen för nftables, bland annat om hur regler ordnas och vilka beslut som avslutar en kedja.
- wg(8): set and retrieve configuration of WireGuard interfaces — WireGuard-manualen, där AllowedIPs definieras som en lista med adresser med CIDR-masker.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är CIDR?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/cidr/