Design-dokument - Ruting

Beskrivelse av rutingsprotokollen, dens oppførsel og hvordan meldingsformatene er satt opp:

Beskrivelse: Rutingsprotokollen er distance vector routing (DVR) over MIP, som er delt mellom mipd og routingd. Routingd oppdager naboer med periodiske HELLO-meldinger hvert sekund, som sendes til alle naboene (MIP-broadcast). Disse blir holdt styr på i en nabo-tabell. I tillegg fører den en rutetabell med feltene (destinasjon, kostnad, neste hopp og alder). Måten rutetabellen oppdateres er med periodiske UPDATE-meldinger, som sendes til hver nabo hvert andre sekund. Mottatte UPDATE-meldinger fra naboer gir et x-antall (dest,kostnad) oppføringer, som sammenlignes med egne oppføringer. For å forhindre løkker benyttes Split Horizon med Poisoned Reverse. Det vil si at når en UPDATE-pakke sendes til nabo n, settes kosten lik «uendelig» for alle destinasjoner der beste rute går gjennom n. Videre er det mipd som håndterer selve videresendingen av pakkene for rutingsprotokollen. Dersom pakken ikke er til selve mipd, vil mipd utføre videresendingen.

- MIP er i seg selv en enkel nettverkslagsprotokoll med en liten header som inneholder kildeadresse, destinasjonsadresse, TTL og type. Etter headeren kommer selve meldingen, altså payloaden fra applikasjonen. Hele pakken sendes direkte inni en Ethernet-ramme og gjør at noder kan sende meldinger til hverandre i nettverket.

Oppførsel: Routingd sender HELLO-melding hvert sekund til alle naboer, og anser en nabo som borte hvis den ikke har mottatt en HELLO-melding innenfor en tidsgrense på tre sekunder. For hver mottatt HELLO-melding, vil naboen bli registrert som nabo, og rutetabellen vil bli oppdatert der naboen er en direkte rute. Dersom en nabo ikke mottar HELLO-meldingen innenfor tidsgrensen markeres naboen som borte. Av den grunn er det viktig å oppdatere alle rutetabellene i topologien om dette. Dette gjør vi ved å sette den direkte ruten til naboen og alle ruter via naboen til kostnad lik uendelig (INF_COST), som blir sendt gjennom en UPDATE med en gang. Dette fører til at de andre rutetabellene oppdaterer rutetabellen sin korrekt. Når UPDATE-pakker blir mottatt vil rutetabellen oppdateres hvis vi mottar en ny destinasjon eller en billigere kostnad. Dersom kostanden er uendelig fra avsender, så invaliderer vi ruter via den (poisoned reverse). I mipd leveres SDU opp hvis dst == min_mip, men dersom pakken ikke skal hit må den videresendes til billigste nabo. Dette gjør mipd ved å sende en request-pakke til routungd, som finner neste hopp fra rutetabellen og svarer tilbake til mipd med en response-pakke. Med response-pakken kan vi sende pakken videre til riktig nabo, som routingd fant for oss. Dersom vi ikke har MAC-adressen for den naboen vil ARP-mekanismen kjøre, og pakken vil bli sendt etter det.

Meldingsformat:

- HELLO: Første byte i denne meldingen forteller mottaker at dette er en hello-melding. Deretter settes tre neste byte til 0 slik at den er fire byte justert. Form: [«H», 0,0,0].
- UPDATE: Meldingen avhenger av N antall oppføringer. Antall oppføringer kommer an på hvor mange elementer det er i rutetabellen. Første byte forteller mottaker at pakken er en UPDATE. Neste byte forteller antall oppføringer som er i pakken (dst, kostnad). Videre er det N oppføringer, hvor hver oppføring er tre bytes stor. Form: [«U», N, (dst, kost),....].

- REQUEST og RESPONSE pakker er fulgt akkurat som oppgaven sier.

Beskriv problemet med «Count-to-Infinity» i DVR protokoller. Hvordan håndterer du dette i din implementering?

I DVR sender hver rute i rutetabellen sin beste kostnad til alle destinasjoner til naboene (UPDATE). Når en lenke eller valgt neste hopp feiler eller forsvinner, ser ikke rutetabellen hele stien, men bare naboens perspektiv. Av den grunn kan to naboer lære en alternativ vei fra hverandre til en destinasjon som fører til en evig loop. De vil øke kostnaden de mottar fra hverandre i hver UPDATE-pakke de mottar. Dette er «Count-to-Infinity» problemet. Måten jeg håndterer det på er med Split Horizon med poisoned reverse. Når jeg bygger en UPDATE-pakke til nabo n, settes alle destinasjoner hvor neste hopp går gjennom n med kostnad lik uendelig (INF_COST) til akkurat den naboen. Videre når jeg mottar UPDATE-pakken sjekker jeg alle oppføringer som har kostand lik INF_COST, og om min nåværende rute til destinasjonen går via naboen som sendte pakken. Dersom dette er sant invalidiseres den naboen fra ruting-tabellen. Dette fører til at jeg aldri går tilbake, slik at det ikke lages en evig loop fram og tilbake (Count-to-Infinity). Med periodiske UPDATE pakker håndteres dette korrekt. I tillegg gjør jeg også en «triggered» UPDATE ved nabotap. Det vil si timeout ved HELLO-pakker. Når det forekommer et nabotap setter jeg alle ruter til INF_COST der neste hopp er lik den «døde» naboen. Deretter sender jeg UPDATE med en gang slik at alle noder får det med seg. Dette fører til at rutetabellene oppdateres korrekt og «Count-to-Infinity» problemet håndteres.

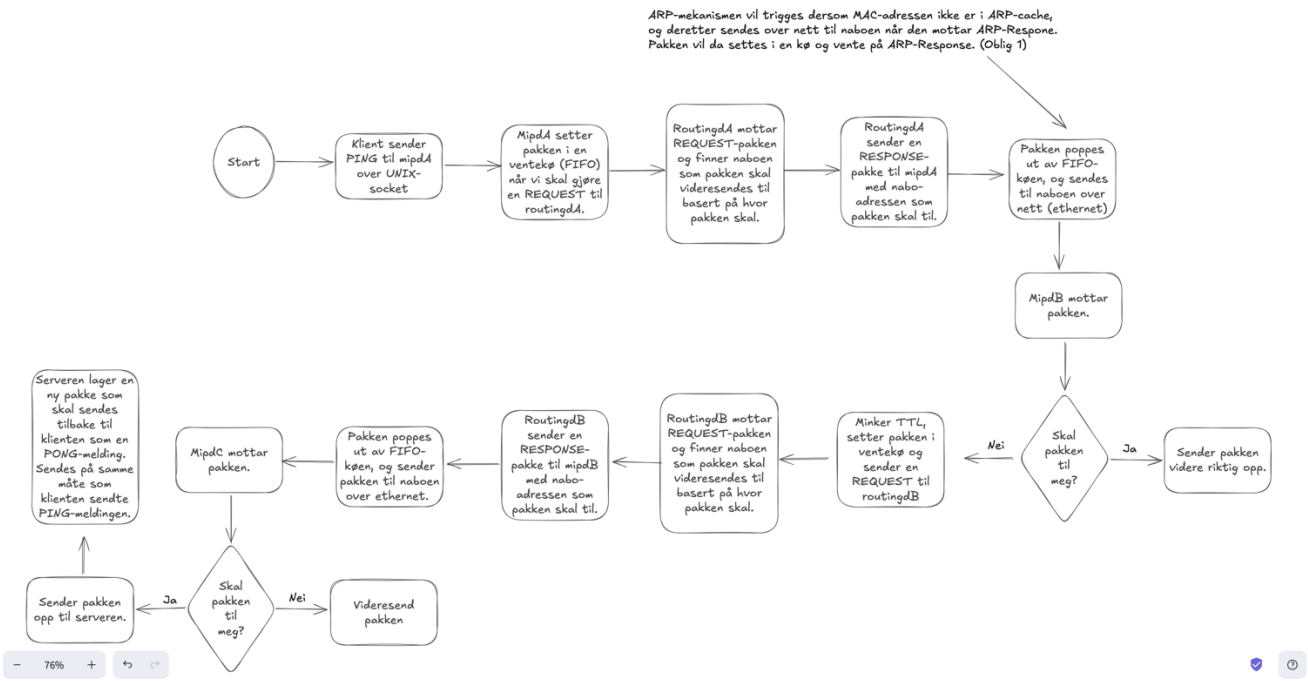
Hva er begrensningen med Split Horizon og hvordan løser Posion Reverse dette problemet?

Split Horizon handler om at man ikke skal fortelle naboen om en vei du lærte av akkurat den naboen (samme link). Det vil si at hvis node B lærte «X er nårbar via A», så sender ikke node B senere en UPDATE til A der B hevder en rute til X. Dette forhindrer to-node-løkkene som er direkte naboer. (A←→B). Begrensningen med Split Horizon er at feil info fortsatt kan sirkulere gjennom andre naboer spesielt når noder har flere naboer. F.eks. B lærer X fra A, C lærer X fra A og når A mister X koblingen, kan B og C fortsatt mate hverandre med «X via deg» i UPDATE-pakkene, som fører til Count-to-Infinity. Poisoned Reverse fikser dette ved at når du sender en UPDATE til akkurat den naboen du går via for en destinasjon, så setter jeg kostnad = uendelig for den destinasjonen til den naboen. Dermed foreller jeg: «jeg når ikke X utenom deg», og naboen vil aldri velge meg som alternativ vei tilbake til seg selv. Med dette fikses begrensningen med Spilt Horizon.

Flytdiagrammer som oppsummerer utførelsesflyten:

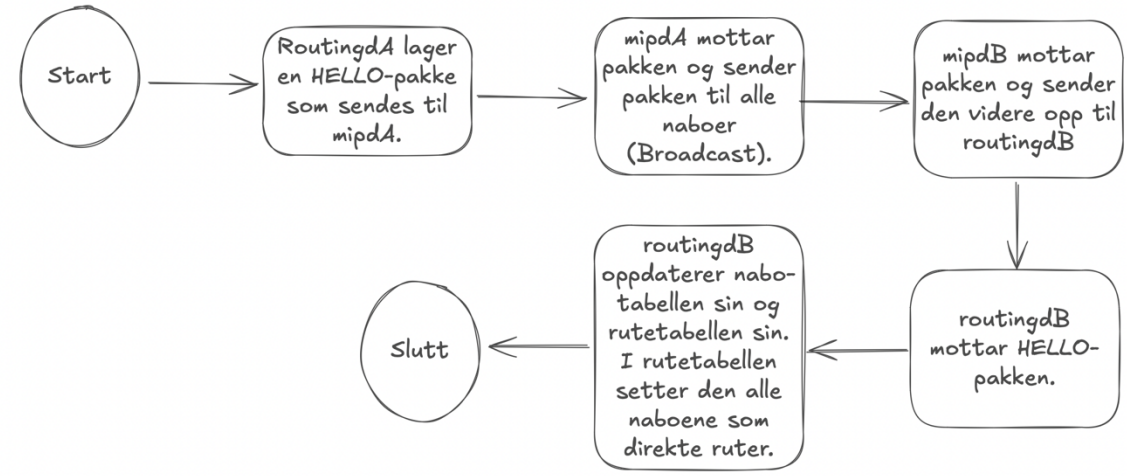
Flytdiagram 1: viser en oppsummering av sekvensen til en klient som skal sende en pakke til en server. Flyten går mellom flere verter. I mitt eksempel har jeg tre verter A - B - C. Hver av vertene har en mipd og routingd. I tillegg har vert A en klient og vert C en server. Klient i vert A skal sende en melding til vert C, hvor server skal motta meldingen.

Flytdiagram 1 (aktivitetsdiagram):



Flytdiagram 2: viser en oppsummering av sekvensen til en HELLO-melding som skal sendes hvert sekund. I vårt eksempel har vi to verter A – B. Hver av vertene har en mipd og routingd.

Flytdiagram 2 (aktivitetsdiagram) :



Flytdiagram 3: viser en oppsummering av sekvensen til en UPDATE-melding som skal sendes hvert andre sekund. I vårt eksempel har vi to verter A – nabo. Hver av vertene har en mipd og routingd.

Flytdiagram 3 (aktivitetsdiagram):