Danmarks Tekniske Universitet

Introduktion til datanet 34313

Simuleringsøvelser

Stefan Artur Stefirta
s244813

Andreas Bundgaard
s244970

Tommy Krogh Andersen
s245569

27. november 2024

1 Indledning

I denne rapport undersøges forskellige principper, som er blevet gennemgået i kurset: 34313 Introduktion til datanet. Rapporten er inddelt i 9 dele, hvor der i hver del udføres simuleringer, for at undersøge de givne principper. De overordnede emner er:

- Flow-control algoritmer (Del 1-3)
- Performance i Ethernet- og Wifi lokalnet (Del 4-6)
- IP-adressering og routing i internettet (Del 7-9)

Formålet med rapporten og simuleringsøvelserne er at få en bredere indsigt i, hvordan kommunikationen fungerer i forskellige typer af datanet. Med denne forståelse kan netværket påvirkes for at demonstrere forskellige karakteristika for hvert datanet, men også at undersøge udfordringerne ved at benytte en specifik teknologi i et netværk. De fleste netværksteknologier har både fordele og ulemper, og ved hjælp af realistiske parametre, kan det være vejledende, hvis et netværk skal konfigureres i virkeligheden.

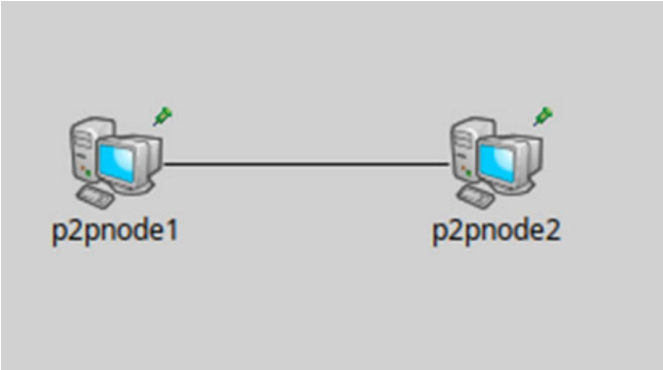
I øvelserne er der et fokus på lokalnet, men nogle af principperne går igen i større datanet. Der reflekteres over hvordan flow-control algoritmer i forskelligt hardware påvirker udnyttelsen af datanettet, og hvilken betydning det kan have for valg af hardware. Ydermere bliver Wifi og Ethernet sammenlignet med henblik på at forklare effektiviteten af de forskellige metoder til at transmittere data over et lokalnet. Slutteligt bliver mindre lokalnet gennemgået ift. IP-adresser og routing på OSI-modellens lag 3, for at analysere, hvordan netværket kan konfigures både statisk og dynamisk. Der bliver undsøgt netværkstrafik for at demonstrere hvordan routere vha. RIP-protokollen selv kan opdatere ruterne under forskellige forudsætninger.

Indhold

2 Del 1-3

Primært udarbejdet af: Tommy Kogh Andersen s245569

Simuleringerne 1-3 omhandler point-to-point protokollen, som netop omhandler en forbindelse mellem to klienter som er direkte forbundet, se topologien, på figur 1 nedenfor. Denne topologi er udgangspunktet for hele del 1-3, hvor den arbejdes med den. Der arbejdes primært med stop-and-wait protokollen, med forskellige parametere, samt kort med go-back-N og slelective-reject protokollerne.



Figur 1: Simpel netværkstopologi for point2point

2.1 Del 1 – Point-to-point protokol med Stop-and-Wait

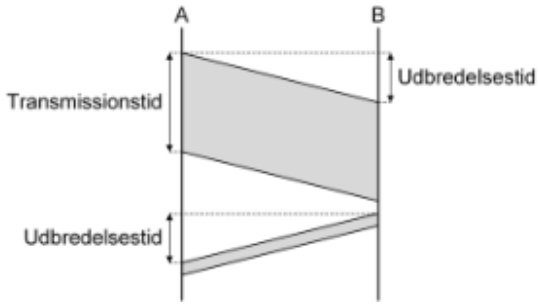
2.1.1 Del 1a - udnyttelse af stop-and-wait

Den første simulation har til formål at undersøge udnyttelsen af point to point protokollen, vha. stop and wait algoritmen. For at forstå betydningen af simuleringen, er det vigigt først analysere konfigurationen.

Section/Key	Value
General	
*.p2pnode1.p2pentity.Rol	2
*.p2pnode2.p2pentity.Rol	3
**.p2pentity.WindowSize	1
**.p2pentity.FrameSize	1024
**.p2pentity.Timeout	30s
**.p2pentity.Verbose	0
**.p2pentity.ARQMode	"Stop-and-wait"
**.channel.delay	1.7s
**.channel.ber	0
**.channel.datarate	1024bps

Figur 2: Konfigaration af point-2-point protokol til del 1a

På figur 2, kan konfigurationen ses, heraf er de vigigste i denne kontekst framesize på 1024 bit, datarate 1024 bit/s , som medfører at at det tager netop $\frac{1024\ bit/s}{1024\ bit} = 1\ s$ at overføre en pakke. Samt at delay(udbredelsestid) på $1.7s$, som er tiden der går fra at afsenderen sender en pakke, til at modtageren, modtager den.



Figur 3: Illustration af informations udveksling[1]

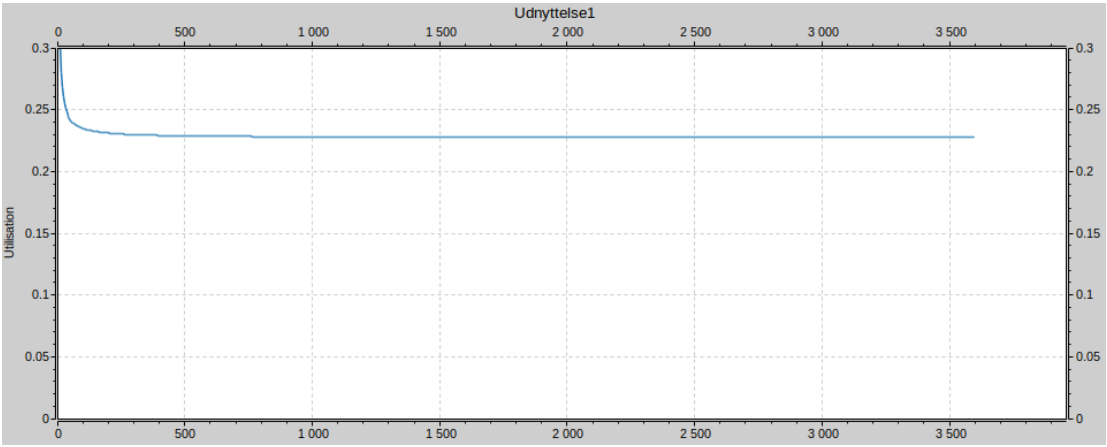
På figur 3 kan det ses at den totale tid, fra afsendelse af meddelse, til bekræftelse er transmitionstid + 2 gange udbredelsestid, ud fra dette, kan den teoretiske udnyttelse findes ud fra formlen.[2]

$$U = \frac{\text{Transmitionstid}}{\text{Total tid}} = \frac{t_{tx}}{t_{tx} + 2t_p} = \frac{1}{1 + 2\frac{t_p}{t_{tx}}}$$

Dvs.

$$U = \frac{1}{1 + 2\frac{1.7s}{1s}} = 0.22727\ldots$$

Det forventes altså, at stop and wait har en udnyttelse på ca. 22.7%.



Figur 4: Udnyttelse af Stop-and-Wait over en time, i 1a

Efter simuleringen, som ud fra konfigurationen kører en time er dataet opsamlet og kan resulteterne ses på figur 4. Ved simuleringens afslutning($3600s$), aflæses udnyttelsen til 0.227484, som netop svarer til 22.7%, ligesom den teoretiske værdi og resultatet af simuleringen er altså, som forventet.

[1] [1] Kapitel 1, afsnit 1.1, Figur 1.1(d)
[2] [1] Kapitel 3, afsnit 2.5.1

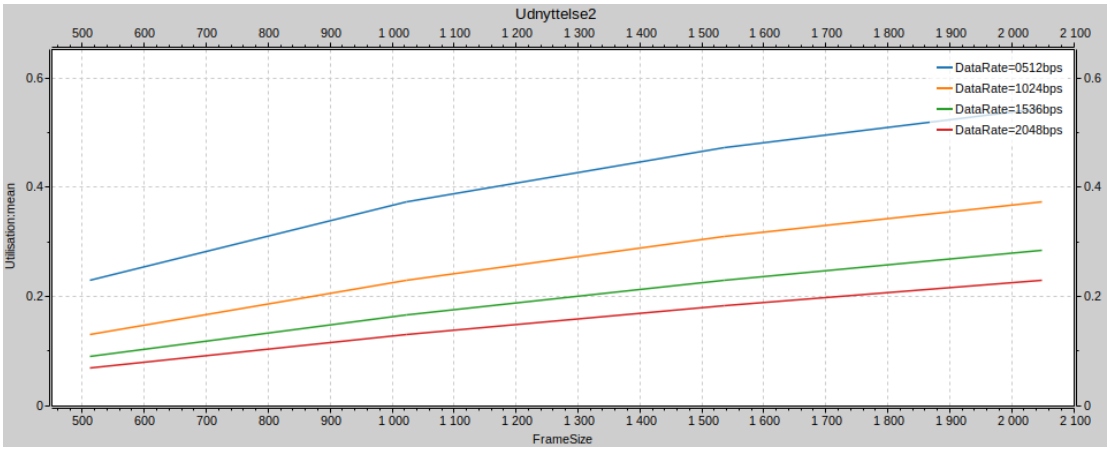
2.1.2 Del 1b - udnyttelse ved forskæellige frames

I denne del ønskes det at undersøge, hvilken indflydelse framesize, samt datarate har på udnyttelsen. for at gøre dette ændres konfigurationen af disse til 4, henholdsvis (512, 1024, 1536, 2048) i både framesize og datarate, se figur 5. På den måde bliver der kørt 16 simuleringer, svarende til de forskællige kombinationer.

Parameter	Ny værdi
**.p2pentity.FrameSize	${FrameSize=512..2048 step 512}
**.channel.datarate	${DataRate=0512bps,1024bps,1536bps,2048bps}

Figur 5: Ædringer i konfiguration(framesize og datarate), fra figur 2, passende til 1b

Hvis der kigges på formlen fra 1a, kan det bemærkes at forlen kun er afhænging af t_{tx}(transmitions tiden), da udbredelsestiden er konstant. Svarende til $t_{tx} = R/L$, hvor R er dataraten og L framesize. Herfra kan det netop afgøres at når dataraten falder, vil den ydre brøk stige dvs. en højere udnyttelse. og omvendt gælder det for framesize.



Figur 6: Udnyttelse af Stop-and-Wait, for 16 configurationer af framsize og datarate (bemærk at der ikke er 'knæk' på grafen, men det blot punkter der er forbundet)

På figur 6, den faktiske udnyttelse efter en time i hver af de 16 simuleringer ses. Her kan det tydeligt ses at udnyttelsen netop er højere for simuleringer med lav datarate. Og at udnytelsen stiger, når framsize er større. Det kan også ses at der er punkter for hver datarate, hvor udnyttelsen ca. er den samme. Netop når den er tilsvarende til framesize og udbredelsetiden er $1s$, svarende til del 1a. Som alt sammen, stemmer overens med det forventede, fra teorien.

2.2 Del 2 – Effektivitet af Stop-and-Wait ved transmissionsfejl

2.2.1 Del 2a - thoughput, ved fejlfri transmitions

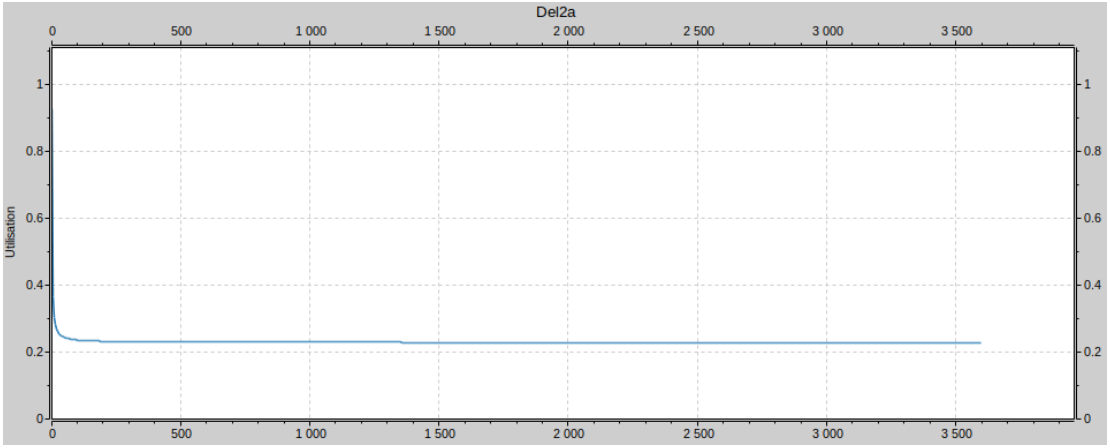
I del 1, har der været arbejdet med udnyttelsen, af Stop-and-Wait, men det er ikke hele billedet af, hvor effektiv algoritmen er. En anden vigtigt faktorer, er thoughput, som er mængden af data, som rent faktisk bliver modtaget. For at undersøge dette bruges samme konfiguration, som i del 1a(se figur 2).

Denne konfiguration, bruges for at eftervise at der er en lineær sammenhæng mellem udnyttelse og thoughput ved en fejlfri transmition, da alt den data der bliver afsendt også modtages. Dette, kan opskrives som.

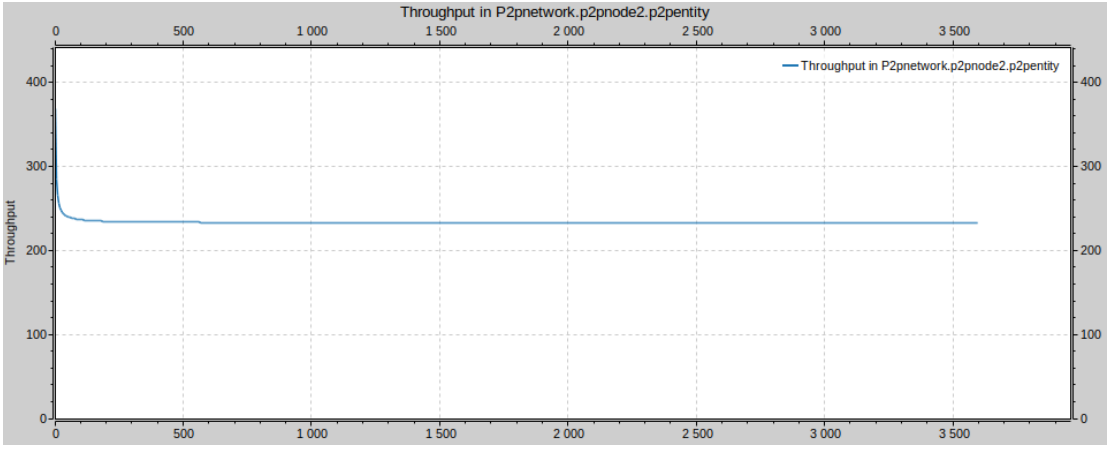
$$T = R \cdot U$$

Hvor T er thoughput, R er dataraten og U er udnytelse. Da der bruges samme konfiguration, fra del 1a. bruges udnyttelsen derfra blot.

$$T = 1024\ bit/s \cdot 0.227 = 232.4\ bit/s$$



Figur 7: Udnyttelse af Stop-and-Wait over en time, i 2a (tilsvarende til figur 4)



Figur 8: Thoughput af Stop-and-Wait over en time, i 2a

På figur 7 kan udnytelsen ses, hvis den sammenlignes med figur 8, kan den lineære sammenhæng ses. Ved aflæsning af thoughput efter $3600s$, findes det til at være $232.8\ bit/s$, som er en nærmest tilsvarende den teoretiske værdi, med en fejlmagin på $1 - \frac{232.8\ bit/s}{232.4\ bit/s} = 0.17\%$, som yderligere fremhæver lineariteten.

2.2.2 Del 2b - thoughput, ved transmitionsfejl

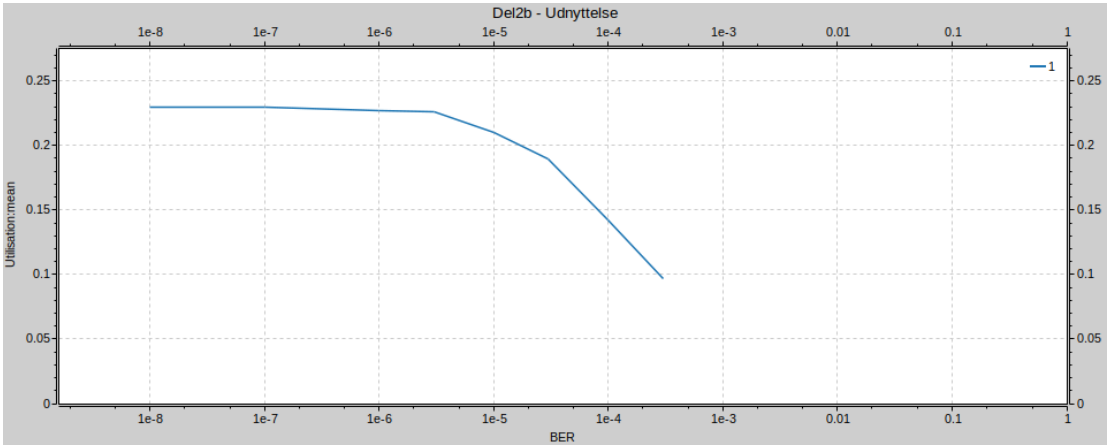
Med udgangspunkt i del 2a, ændres konfigurationen. Således at der foretages 8 simuleringer, med forskællig transmitionsfejls sansynlighed(se figur 9). På den måde kan det undersøges hvilken indflydelse dette har på henholdsvis udnyttelse og thoughput. Det er også vigtigt at pointere at timeoutværdien er på $30s$ (se figur 2).

Parameter	Ny værdi
`**.channel.ber`	`${BER=1e-8,1e-7,1e-6,3e-6,1e-5,3e-5,1e-4,3e-4}`

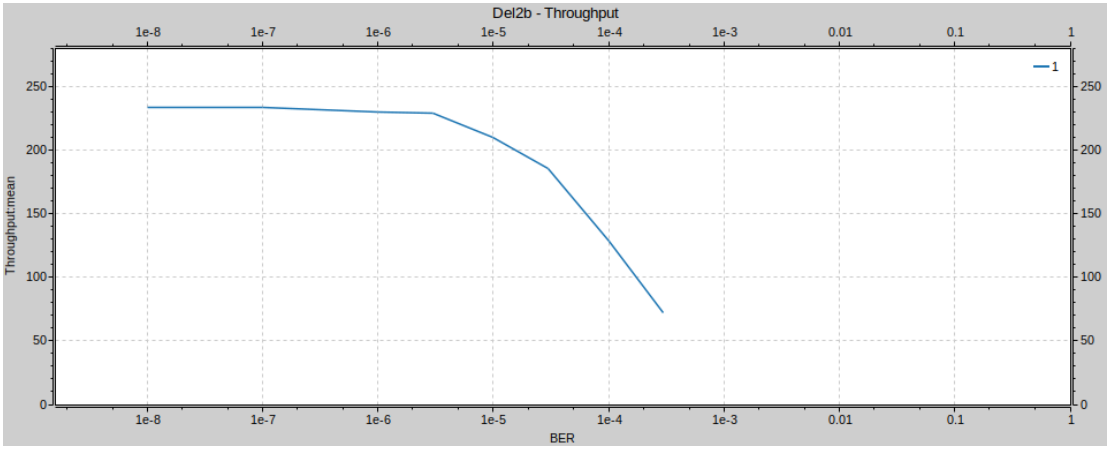
Figur 9: Ændring i konfiguration, så der foretages 8 simuleringer med stigende sansynlighed for transmitionfejl

Hvis der sker en transmitionsfejl, vil modtageren smidde pakken væk, og ikke sende en bekræftelse(ack) til afsenderen. Dette medfører at afsendere, må vente til timeoutværdien($30s$) er gået og retransmitere pakken. Som konsekvens af dette, vil der gå længere tid mellem hver transmition, da der skal ventes længere en den originale forsinkelse på $2 \cdot 1.7s = 3.4s$.

Da thoughput var defineret ved modtagede pakker, vil det være lavere end udnyttelsen. Dette skyldes at den netop ikke kan bruge pakkerne med fejl, så de bliver ikke talt med, selvom de er en del af udnyttelsen.



Figur 10: Udnytelsen af Stop-and-wait, med 8 forskællige transmitionsfejl sansynligheder, i 2b (bemærk logaritmisk x-akse)



Figur 11: Thoughput af Stop-and-wait, med 8 forskællige transmitionsfejl sansynligheder, i 2b (bemærk logaritmisk x-akse)

På figur 10, kan udnyttelsen ses og tilsvarende kan thoughputtet ses på figur 11. Som forventet, kan det ses at thoughputtet falder mere relativit til udnyttelsen, som forventent. Grunden til at udnyttelsen begynder at falde drastisk er grundet at timeoutværdien er en konstant, som netop er væsenligt større end forsinkelsen. Som medfører at den har større indfydelse, ved flere fejl.

2.3 Del 3 – Effektivitet af Sliding-Window flow control

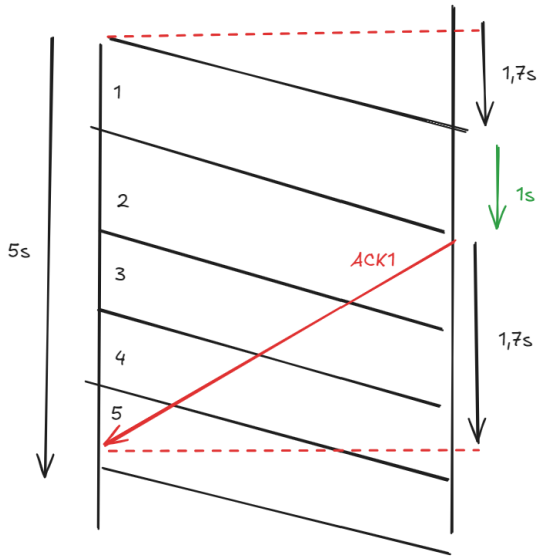
2.3.1 Del 3a - varierende windowsize

I denne del, vil der være fokus på hvad udnyttelsen, af point to point protokollen fungere, når der gøres brug af sliding window(se figur 12 til flow control. Dette gøres stadig vha. stop and wait, uden transmitionsfejl.

Parameter	Ny værdi
**.p2pentity.WindowSize	${WindowSize=1..7}

Figur 12: Ændring i konfigurationen, så der simuleres for vinduestørelse 1-7

Når der gøres brug af sliding window, betyder det at den kan sende n pakker, før den behøver at vente på en ack, fra den først sente, som gør udnyttelsen væsentligt højere. Dette medfører en direkte sammenhæng, mellem udnyttelsen og vinduestørelsen dvs. $U \cdot n$, hvor n er windowsize. Dette gælder dog ikke altid, da udnyttelsen selvfølgelig ikke kan være over 1.

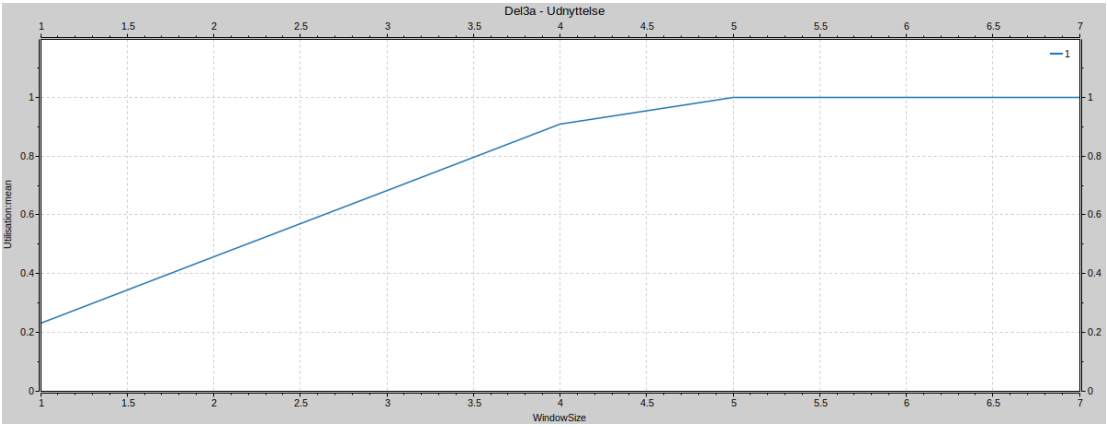


Figur 13: Skitsering af framesize på 5 eller større

For bedre at forstå hvornår udnyttelsen vil være 1, er det skitseret på figur 13, kan det ses at unyttelsen vil være 1, da $5 \cdot 1s > 1s + 2 \cdot 1.7s$, så den vil have modtaget en ack, før den er færdig med at sende den sidste medelse i vinduet. Derfor kan formlen omskives til.

$$f(U,n) = \begin{cases} U \cdot n, & U \cdot n < 1 \\ 1, & U \cdot n \geq 1 \end{cases}$$

Dvs. at udnyttelsen nu kan beregenes til for alle windowssize, såfremt der ikke er transmitionsfejl.



Figur 14: Udnyttelsen af Stop-and-Wait, med vindustørelse 1-7

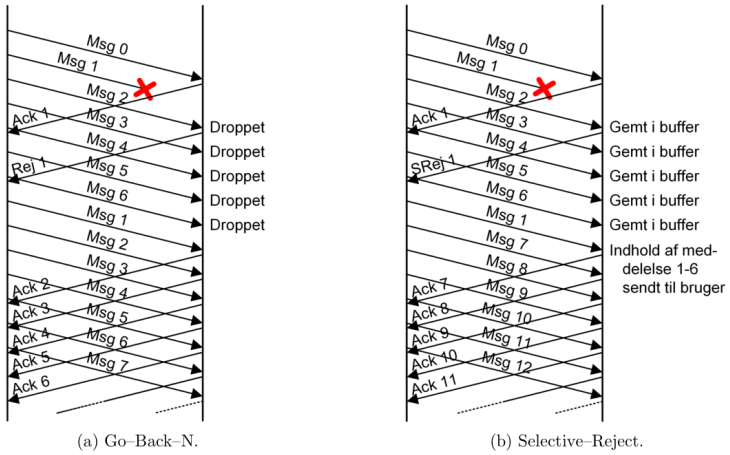
På figur 14, kan udnyttelsen af ses for windowsize 1-7, og det kan ses at den som forventet stiger lienært, indtil den rammer 1. Principielt ville udnyttelsen ramme en mellem 4 og 5, men da windowsize, skal være et helt antal, er linjen derimellem er der det ekstra knæk.

2.3.2 Del 3b - Go-back-N og Selective-Reject

I denne del kigges der på forskellen af thoughputtet, ved brug af Go-back-N og Selective-Reject, med forskellige transmitionsfejl sansynligheder(se figur 15).

Parameter	Ny værdi
**.p2pentity.WindowSize	7
**.channel.ber	${BER=1e-7,3e-7,1e-6,3e-6,1e-5,3e-5,1e-4,3e-4,1e-3}
**.p2pentity.ARQMode	${ARQ="Go-back-N","Selective-Reject"}

Figur 15: Ændring i konfigurationen, på vinduestørelse, transmitionsfejl samt ARQ-principper



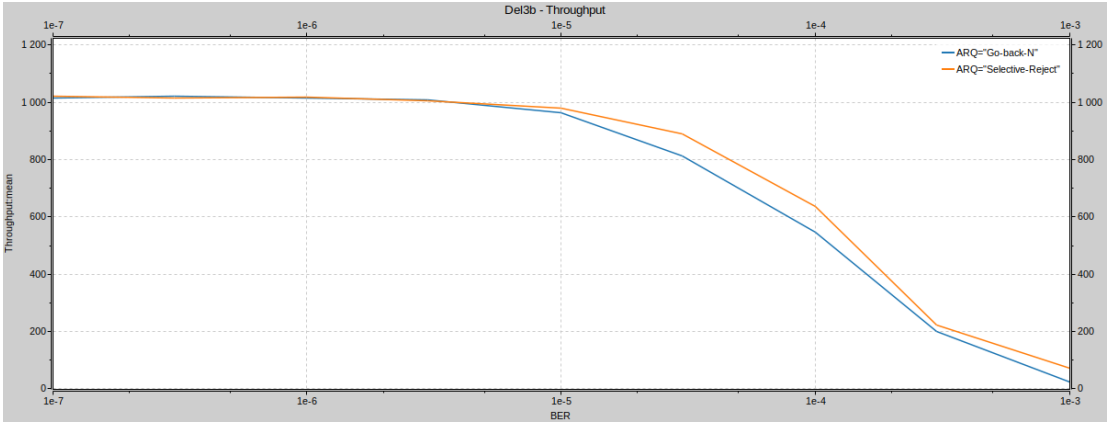
(a) Go–Back–N. (b) Selective–Reject.

Figur 16: Go–Back–N og Selective–Reject med "stor" vinduesstørrelse.[3]

[3] [1] Kapitel 3, afsnit 2.7.3, Figur 3.17 (a) og (b)

På figur 16, kan forskællen mellem de to principper ses. Når Go-back-N opdager en fejl i de modtagede pakker, gives en reject meddelse og afsendere, vil derefter gensende alle medelser fra og med den med fejl, så modtageren får alle pakker fejlfrit. Problematikken er netop at en masse pakker bliver droppet.

Modsat gør Selective-Reject brug af en buffer, som gemmer alle pakker inden for vinduet, så en reject meddelse vil medføre at den pakke med fejl, skal gensendes. Det forventes derfor at Selective-Reject har et højere thoughput, end Go-back-N.



Figur 17: Thoughput af Go-back-N og Selective-Reject, med forskellige transmitionsfejl sansynligheder

På figur 17, kan det ses, at thoughputtet for begge principper er nærmst det samme ved lave transmitionsfejl. Men som forventet er Selective-Reject bedere end Go-back-N når fejlsandsynligheden stiger. Grunden til at thoughputtet begynder at flade ud, når fejlsandsynligheden er ekstremt høj, er at den går mod nul, men ikke kan fortsætte i samme tempo. Dette er dog ikke relevandt dan systemer med så høj fejlsandsynlighed ikke er brugbare i praksis.

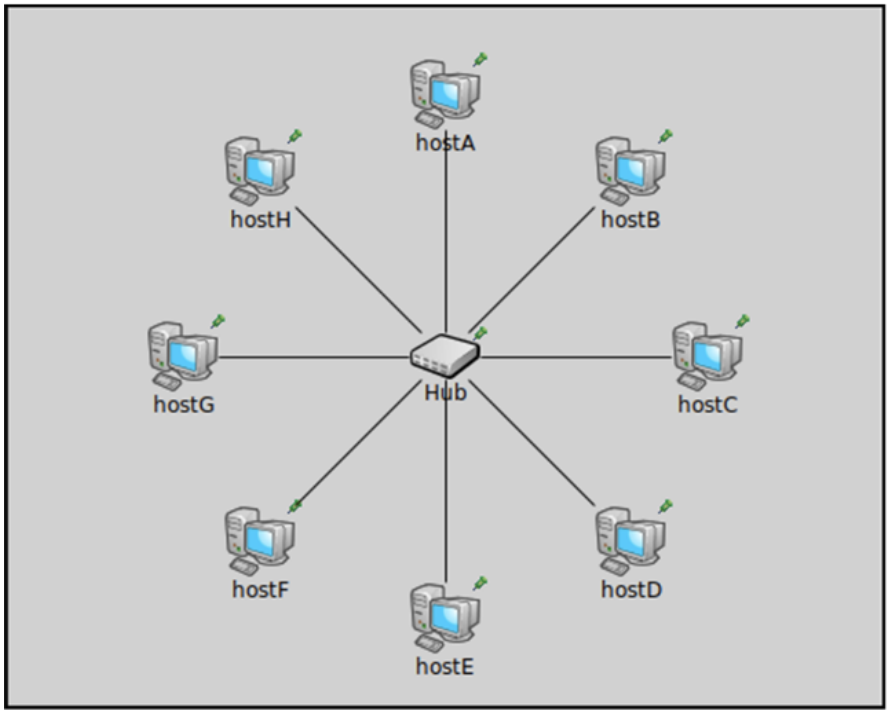
Kigges der på kun på grafen i en implementerings sammennhæng, virker det åbenlyst at gøre brug af Selective-Reject, da den altid har et større thoughput og dermed er mere effiktiv. Dog er det ikke så simpelt, da Selective-Reject, som nævnt gør brug af en buffer som betyder at der vil være større omkostininger i forbindelse med implementeringen af systemet og sansynligtvis mere vedligehold. Det vil derfor i mange ikke kritistske sammenhænge være smartere at gøre brug af Go-back-N.

3 Del 4-6

Primært udarbejdet af: Stefan Artur Stefirta s244813

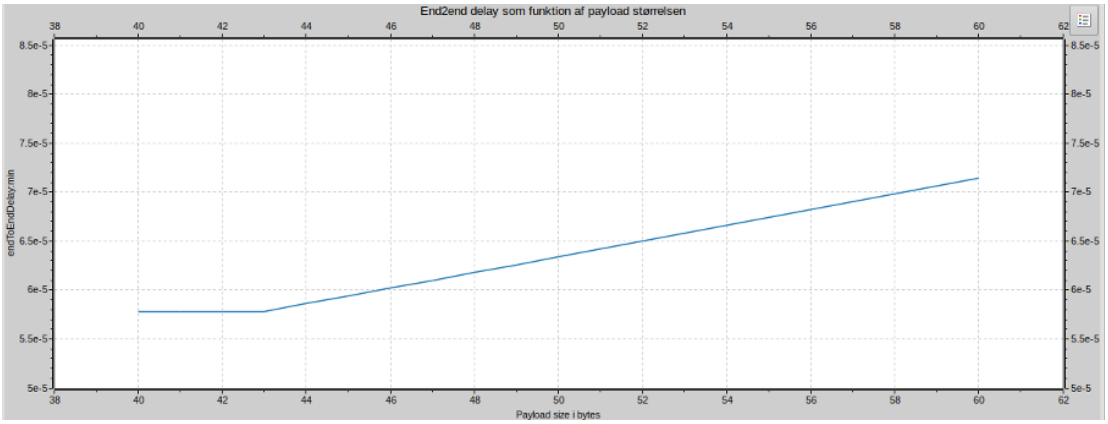
Simuleringenerne 4-6 omhandler Local Area Networks(LANs) og primært hvordan forsinkelsen af transmission af data over en hub, switch eller WiFi forbindelse fungerer. Der vil blive gået i dybde med forskellen på de forskellige netværksenheder og hvorfor samt hvordan deres forsinkelser er forskellige over end-to-end forbindelser.

3.1 Del 4 - End-to-end delay i Hub-baseret LAN



Figur 18: Topologi over Hub-baserede LAN

Ovenfor ses topologien for det hub-baserede LAN som repræsenterer en stjerne-topologi. Simulationen fungerer således at hub'en broadcaster ud til alle subnettets terminaler undtaget afsender terminal, hvor der er en samme konstant forsinkelse mellem hver forbindelse. Formålet med denne delopgave er at måle end-to-end forsinkelser, altså forsinkelsen af pakkerne i datatransmissionens forsendelse fra terminal til slut-terminal, over hub forbindelse i et LAN. Dette udføres ved hjælp af parametrene cli.sendInterval og cli.reqLength som står for tidsintervallet mellem meddelelser og længden af brugerens meddelelser i simuleringen. Nedenfor er en graf som repræsenterer vores resultater af end-to-end forsinkelse i forhold til payload size i bytes. Ud fra vores resultater kan vi udregne hældningen som forklarer forholdet mellem delta cli.reqLength og delta end-to-end forsinkelsen.



Figur 19: Graf over end-to-end forsinkelsen

For at finde hældningen, vi er interesserede i, udregnes højre lineære graf for knækket.

$$a = \frac{\Delta y}{\Delta x}$$

Vi aflæser to vilkårlige punkter på højreside grafen således at vi kan finde hver især x og y differensen, altså delta, som vi kan indsætte i formlen.

$$P_1(46; 0,0000602)$$
$$P_2(50; 0,0000634)$$
$$a = \frac{0,0000634 - 0,0000602}{50,0 - 46,0} = 8,00 \cdot 10^{-7} \frac{s}{\text{byte}}$$

Vi konverterer bytes til bits.

$$\frac{a}{8} = 8,00 \cdot 10^{-7} \cdot \frac{1}{8} = 1,00 \cdot 10^{-7} \cdot \frac{s}{\text{bit}}$$

Vi tager den reciprokke værdi af hældningen, da hældningen er den reciprokke af bitraten.

$$R = \left(1,00 \cdot 10^{-7} \cdot \frac{s}{bit}\right)^{-1} = \frac{1}{10^{-7}} \frac{\text{bit}}{\text{s}} \rightarrow 10^7 \frac{\text{bit}}{\text{s}} = 10 \frac{\text{Mbit}}{\text{s}}$$

Nu er vi sikre på at udregningen er rigtig, da bitraten stemmer.

Vi kan derfor se, ud fra udregningen, at hældningen til højre for knækket er R^{-1}. I udregningen har vi aflæst to vilkårlige punkter, hvor grafen er lineær. Ved at tage den reciprokke værdi af hældningen her, får vi altså bitrate R, som i opgaven blev oplyst til at være 10 Mbit/s. Den konstante aflæste værdi af MAC-ramme længde fra 40-43 (venstre graf for knækket) skyldes MAC-padding som tilføjes til rammen. Hvis user payload ikke er af betydelig størrelse, bliver der tilføjet padding, som sørger for at man har en minimumsstørrelse af pakker i forbindelsen. Dette er vigtigt, da hvis en pakke er for lille, kan den fuldt ud sendes, før en eventuel kollision bliver opdaget, hvilket vil forårsage problemer. Dette betyder, at CSMA/CD ikke kan fungere korrekt. Grafen viser, at der fra 43 bytes i frame size ikke bliver tilføjet padding. Man kunne forvente 46 bytes normalt, men vores Logical Link Controller(LLC) header som fylder 3 bytes

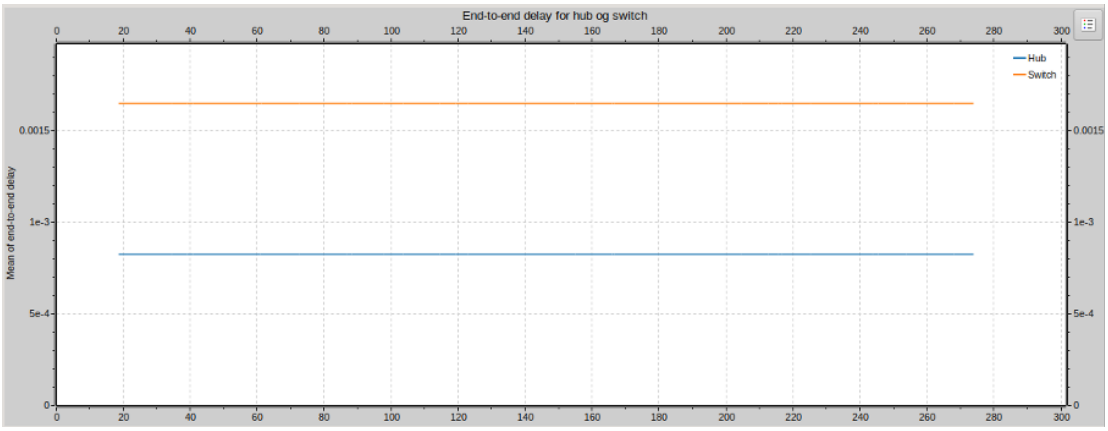
forklarer, hvorfor at frame size først vokser efter 43. Da der nu kun skal 43 bytes af payload data, til at udfylde minimumsstørrelsen på MAC-rammen uden padding. Det bemærkes at L i formlen nedenfor beskriver længden af MAC-rammen i antal bytes.

$$L - LLC_{hdr} = 46\ bytes - 3\ bytes = 43\ bytes = payload_{min}$$

Da der er tale om et kablet netværk, altså ikke en trådløs forbindelse, over ethernet, kan vi fokusere på CSMA/CD(Carrier Sense Multiple Access with Collision Detection), og det er derfor, der bliver tilføjet padding. CSMA/CD(kollisionsdetektion) fungerer altså sådan, at hvis en ramme er mindre end 46 bytes, tilføjes ekstra padding for at opnå den nødvendige minimumsstørrelse på MAC-rammen. Det kan også bemærkes, at forbindelsen er halv-dupleks, hvilket er grunden til, at kollisionsdetektion implementeres. I en fuld-dupleks forbindelse kan der ikke opstå kollisioner, og derfor er kollisionsdetektion ikke nødvendig, hvormed padding stadig tilføjes. kollisionsdetektionen er altså nødvendig for hub baserede lokalnet, hvor alle enheder deler samme kommunikationskanal og dermed kan forårsage kollisioner.

3.2 Del 5 - End-to-end delay i hub, switch og WiFi LANs (WLAN)

I denne næste del implementeres en switch som netværksenhed hvorledes netværkstopologien er den identiske med den i del 4. Nu kan de to LANs sammenlignes i simuleringen, hvor der aflæses resultater baseret på end-to-end delay over tid i simuleringen. Dette er vist på figur 20 nedenfor som repræsenterer vores resultater.

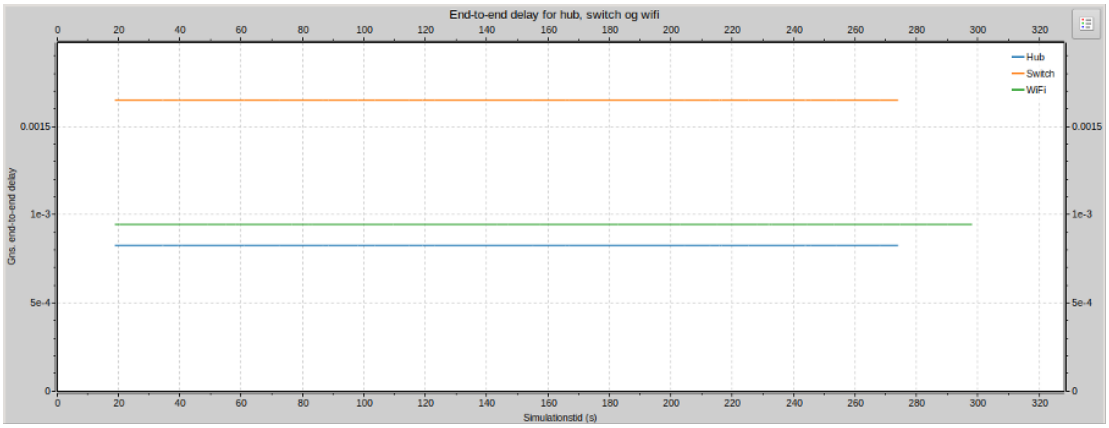


Figur 20: Graf over end-to-end delay, for hub og switch

Når vi kører simulationen, og ser på animationen, kan vi se, at inden switchen sender en besked skal den "flood'e" altså broadcaste beskeden over lag 2, for at få etableret en MAC-tabel som midliertidigt bliver gemt i switchens buffer. Dvs. at data skal sendes frem og tilbage over forbindelsen før at switchen sender data til de rigtige modtagere, da switchen skal vide hvilke porte dataen skal sendes over. En hub bruger ikke lige så lang tid på at sende data over et netværk i modsætning til en switch, da en hub fungerer mere som en repeater og altid videresender data fra én enkelt port til alle modtager porte, altså undtaget afsender port, i sit subnet. Dette forklarer altså hvorfor hub-baserede netværk har en forsinkelse på 0,8ms mens det switch-baserede netværk næsten har en forsinkelse på det dobbelte. Det skal bemærkes at denne forsinkelse sker i det at man starter simulationen fordi at switchen først skal etablere sin MAC-tabel, hvorimod mens MAC-tabellen allerede er blevet oprettet, er der ikke forskel på forsendelsestiden, hvilket

forklarer hvorfor graferne har forskellig startsværdi men dermed begge er konstante værdier for resten af simulationstiden.

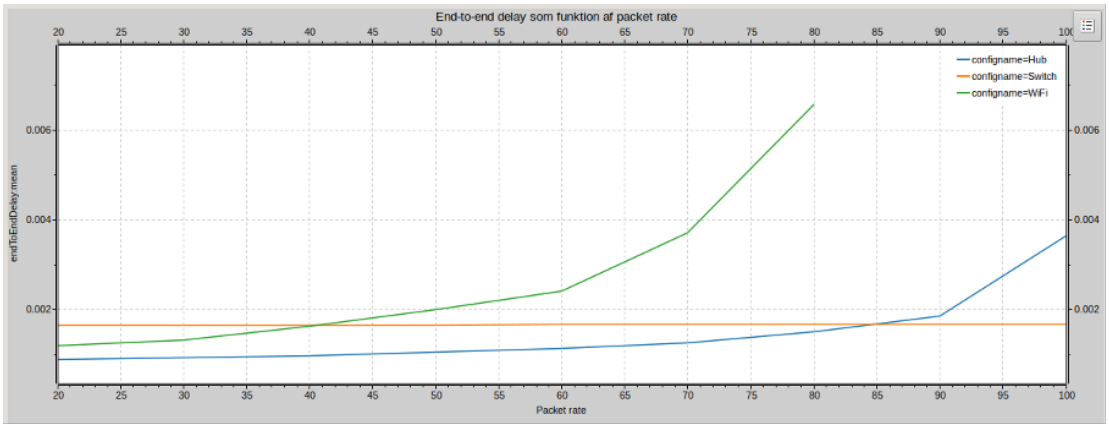
Nedenfor på figur 21 sammenlignes der nu også med et wifi accesspunkt som er blevet placeret istedet for hub'en og hvorledes datatransmissionen nu foregår over WiFi istedet for ethernet.



Figur 21: Graf over end-to-end delay for hub, switch og WiFi

Som det kan aflæses på grafen ovenfor, er forsinkelsen for Wifi nærmere på forsinkelsen af det hub-baserede netværks forsinkelse end det switch-baserede.

Når vi ser simulations-animationen for wifi, kan vi se, at når en terminal vil sende noget, broadcaster den beskeden ud til alle, og kun den relevante modtager accepterer beskeden. Det er nogenlunde samme adfærd som hubben, da beskeden ikke skal igennem et intermediary device, som skal lave en MAC-tabel. Det skal dog bemærkes, at Wifi-forbindelsen har en højere bitrate, L, end de kablede forbindelser. Proceduren for transmission af beskeder for hhv. hub og wifi, minder om hinanden, men der er forskel på det fysiske medium (Lag 1), altså trådløst og kablet, samt MAC-algoritmen, (Lag 2). WiFi bruger CSMA/CA collision avoidance. Denne metode benytter ikke padding, i modsætning til CSMA/CD ethernet, men derimod et DIFS og et SIFS interval til at hellere undgå kollisioner fordi det ikke er muligt at detektere kollisioner over wifi. DCF Interframe Spacing(DIFS) og Short Interframe Spacing(SIFS) benyttes af WiFi ved at terminalen lytter til netværket før den sender over WiFi. Der ventes altså et tidsinterval af DIFS som f.eks. kan være på 3 timeslots inden der sendes data fra terminalen. Derefter skal der ventes et tidsinterval af SIFS som er 1 timeslot, kaldet backoff-intervallet, som udsendes hvis nu at samme terminal skal sende yderligere data. Et timeslot unit skal svare til en længere tid end den transmissionstid fra terminal til terminal-ende over netværkets medium, da det skal være sikkert at timeslottet gælder for alle terminaler over netværket. Herunder er det også vigtigt at pointere at tidsintevallet er meget kort, eksempelvis på få mikrosekunder.



Figur 22: graf over end-to-end forsinkelsen som funktion af belastningen i de tre netværk

Figur 22 for oven illustrerer hvordan end-to-end forsinkelsen ændres som funktion af trafikbelastningen i netværk som er baseret på forskellige enhedstyper: hubs, switches og WiFi. Forskellene i forsinkelse kan forklares ud fra, hvordan disse enheder håndterer trafik under stigende belastning.

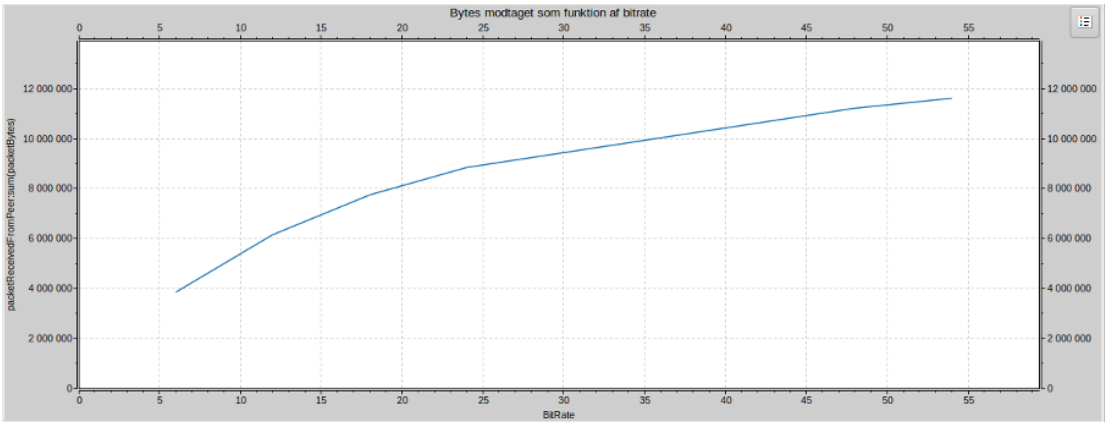
En hub videresender data til alle tilsluttede enheder undtagen den, der sendte beskeden. Når netværket så er belastet med packet-rate, vil det at den broadcaster føre til mere ineffektivitet, da al trafik skal behandles af alle enheder. Dette resulterer i en markant stigning i forsinkelsen, efterhånden som packet-raten stiger, hvilket gør hubs mindre egnede til netværk med stor trafikmængde.

En switch derimod opererer mere effektivt, da den lærer, hvilke enheder der er forbundet til hvilke porte ved at oprette en MAC-tabel, og dermed sender data direkte til den relevante modtager. Denne målrettede dataoverførsel reducerer den unødvendige trafik, som hubs ellers genererer ved at broadcaste, og gør det muligt at håndtere flere samtidige transmissioner, hvilket gør en switch bedre egnet til et LAN med højere packet-rate.

Wi-Fi fungerer på en måde, der ligner en hub, da det broadcaster data til alle tilsluttede enheder. Forskellen her er dog, at Wi-Fi ikke har evnen til at opdage kollisioner, i modsætning til et kablet ethernet netværk. I stedet benyttes CSMA/CA collision avoidance mekanismerne, som forklaret tidligere, til at regulere adgangen til netværket og sikre, at kun en enhed sender ad gangen. Selvom disse teknologier er nødvendige for at undgå kollisioner i trådløse netværk, skaber de yderligere forsinkelser, især når trafikmængden stiger. Dette betyder, at WiFi under høj belastning kan opleve en større stigning i forsinkelse end både hubs og switches.

Generelt viser analysen, at switches håndterer høje trafikmængder bedst ved at reducere unødvendig trafik og tillade flere samtidige transmissioner. Hubs og WiFi, som begge benytter broadcast-metoder, skaber større forsinkelser ved høj belastning, hvilket gør switches til den mest effektive løsning i netværk, der skal håndtere store datamængder.

3.3 Del 6 - WiFi throughput som funktion af den fysiske bitrate



Figur 23: Graf over throughput som funktion af BitRate

Figur 23 for oven forklarer vores resultat af forholdet mellem bitraten og throughputtet. Vores beregninger til grafen udgør således:

Value (bytes)	BitRate (Mbit/s)	Throughput (Mbit/s)
3,864,708	6	2.06
6,168,470	12	3.29
7,770,342	18	4.13
8,850,800	24	4.73
11,219,238	48	5.98
11,623,868	54	6.21

Tabel 1: Value, BitRate og Throughput (Mbit/s)

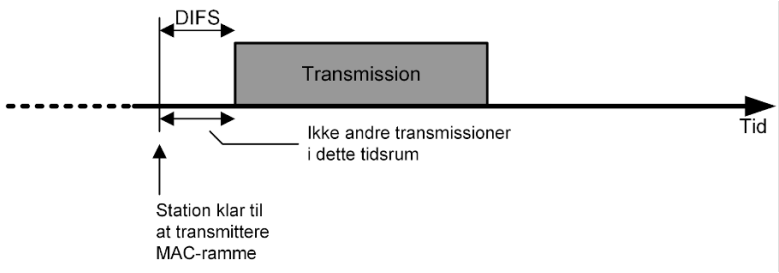
Ud fra beregningerne beskrives sammenhængen ved formlen, hvor V er Value, T er throughput og t er simulationstiden, som i opgavenbeskrivelsen var beskrevet til at være på 15 sekunder:

$$T = \frac{\mathrm{V} \cdot 8}{\mathrm{t}}$$

Ved denne formel er throughput dog beregnet i bit/s, men i tabellen har man divideret med $1 \cdot 10^6$ for at konvertere det til Mbit/s.

Throughput (Mbit/s) er typisk lavere end den egentlige BitRate (Mbit/s). Dette skyldes blandt andet overhead samt DIFS på 50 µs, der påvirker transmissionstiden. Throughput beskriver, hvor meget af den nyttige payload-data der faktisk når frem til modtageren, mens bitrate refererer til al data, der sendes fra afsender til modtager, inklusive overhead.

Som forklaret tidligere i del 1-3, udtrykkes forholdet mellem throughput og bitrate som $T = R \cdot U$, hvor R er bitraten, og U er udnyttelsesgraden. Grafen for throughput stiger dog ikke lineært med bitraten, fordi DIFS forsinkelsen, får mindre betydning forholdsvis i takt med, at bitraten øges. Dette skyldes, at bitraten er en variabel, mens DIFS forbliver uændret i takt med forøgelsen af bitraten. Derfor bliver DIFS' indflydelse mindre i forhold til den samlede transmissionstid, når bitraten bliver højere.



Figur 24: Kilde: (Figur 4.22a, fra "Introduktion til datanet" [1])

Den anden faktor er headerstørrelsen, som også er konstant. Ved højere bitrate tager det mindre tid at transmittere headeren, hvilket yderligere reducerer overheadets indflydelse på throughput. Dette betyder, at stigningen i throughput flader mere ud, jo højere bitraten bliver, fordi effekten af en marginal forøgelse i bitraten mindskes.

Sidst kan det bemærkes, at grafen for throughput har små "knæk". Disse opstår ikke på grund af en egentlig uregelmæssighed, men skyldes, at data i simulationen kun aflæses på specifikke punkter.

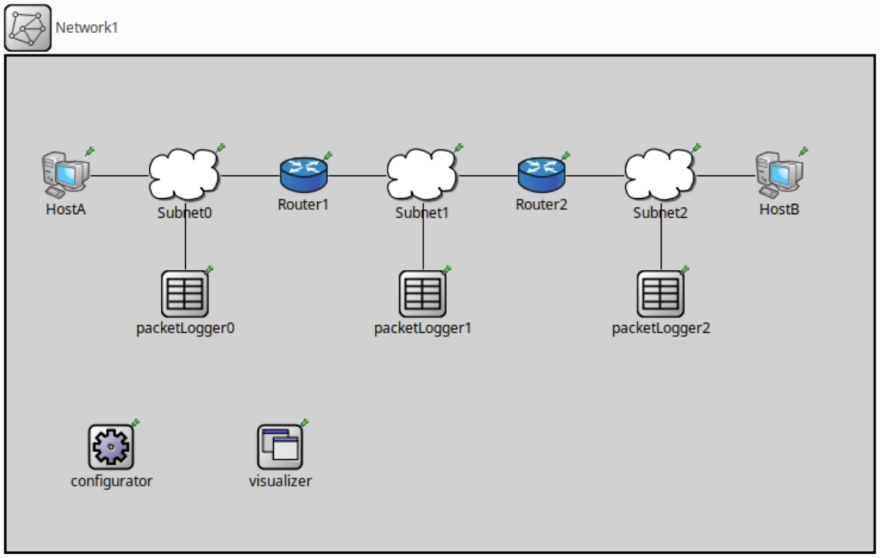
4 Del 7-9

Primært udarbejdet af: Andreas Bundgaard, s244970

Simuleringenerne 7-9 omhandler IP-routing i internettet. Her bliver blandt andet undersøgt hvordan et netværk kan konfigureres statisk og hvilken indflydelse forkerte konfigurationer har på et netværk. Der vil blive gået i dybden med statisk og dynamisk routing.

4.1 Del 7 - Statisk konfiguration af mindre IP-baseret netværk

I denne del af øvelsen bliver et mindre lokalnet konfigureret fra bunden med IPv4. Ved korrekt konfiguration, skulle der gerne opnås en end-to-end forbindelse mellem HostA og HostB. Hvert subnet er udstyret med en packerLogger, som sniffer netværkstrafikkemn med henblik på videre analyse.



Figur 25: Network 1 - Topologi til del 7

På Figur 25 ses hvordan nettet mellem HostA og HostB er struktureret. HostA og B repræsenterer to desktop computere, hvormed skyen repræsenterer et subnet, altså det kan være bestående af switch, hub eller et access-punkt eller en kombination af alle i et subnet. Kort sagt: intermediary devices, der virker på OSI-modellens lag 2. Hvert subnet er forbundet til en router, som sørger for at en end-to-end forbindelse kan opnås.

```
Node Network1.HostA
-- Routing table --
Destination   Netmask       Gateway       Iface               Metric
192.168.0.0   255.255.255.0 *             eth0 (192.168.0.2)      2
127.0.0.0     255.0.0.0     *             lo0  (127.0.0.1)        1
*             *             192.168.0.1   eth0 (192.168.0.2)      0

Node Network1.HostB
-- Routing table --
Destination   Netmask       Gateway       Iface               Metric
192.168.2.0   255.255.255.0 *             eth0 (192.168.2.2)      2
127.0.0.0     255.0.0.0     *             lo0  (127.0.0.1)        1
*             *             192.168.2.1   eth0 (192.168.2.2)      0

Node Network1.Router1
-- Routing table --
Destination   Netmask       Gateway       Iface               Metric
192.168.0.0   255.255.255.0 *             eth0 (192.168.0.1)      2
192.168.1.0   255.255.255.0 *             eth1 (192.168.1.1)      2
127.0.0.0     255.0.0.0     *             lo0  (127.0.0.1)        1
*             *             192.168.1.2   eth1 (192.168.1.1)      0

Node Network1.Router2
-- Routing table --
Destination   Netmask       Gateway       Iface               Metric
192.168.1.0   255.255.255.0 *             eth0 (192.168.1.2)      2
192.168.2.0   255.255.255.0 *             eth1 (192.168.2.1)      2
127.0.0.0     255.0.0.0     *             lo0  (127.0.0.1)        1
*             *             192.168.1.1   eth0 (192.168.1.2)      0
```

Figur 26: Network 1 - konfiguration

Som ses af Figur 26 er der forskellige parametre hvormed hver node er konfigureret. Alle netværk har CIDR /24 notation, som ses ved at alle netmasks har værdien 255.255.255.0. Dette har vi selv konfigureret. Gateway parameteren er vores default gateway, som er den destination hver node vil sende sin pakke hen, hvis den ikke selv kender ruten. Default gateway er en router, der er forbundet, eller kender vejen til den pågældende destination. Subnet0 har IP'en: 192.168.0.0, Subnet1 har: 192.168.1.0, og Subnet2 har 192.168.2.0. HostA har ét interface forbundet i Subnet0, Router1 har to interfaces[4], som hhv. er forbundet i Subnet0 og i Subnet1, Router2 et interface forbundet i Subnet1 og et i Subnet2, og slutteligt har HostB ét interface forbundet i Subnet2.

```
No.         Time          Source        Destination    Protocol Length Info
    1 300.000000    192.168.0.2   192.168.2.2    ICMP         102        Echo (ping) request  id=0x0000, seq=0/0, ttl=32

No.         Time          Source        Destination    Protocol Length Info
    2 300.000005    192.168.2.2   192.168.0.2    ICMP         102        Echo (ping) reply    id=0x0000, seq=0/0, ttl=30
```

Figur 27: packetLogger0 capture data

Som ses af Figur 27, forekom der en succesfuld ping transmission ved at afvikle simulationen. Relevante dele af headeren er markeret med rødt. For at en ping-transmission anses som succesfuld, skal en request kvitteres med en reply. Reply SKAL også nå tilbage til modtageren. Netværket er blevet konfigureret på en måde, så både icmp request og icmp reply blev opsnappet i Subnet0, hvilket må betyde, at konfigurationen er korrekt. ICMP pakkerne er altså nået fra afsender til modtager og tilbage igen.

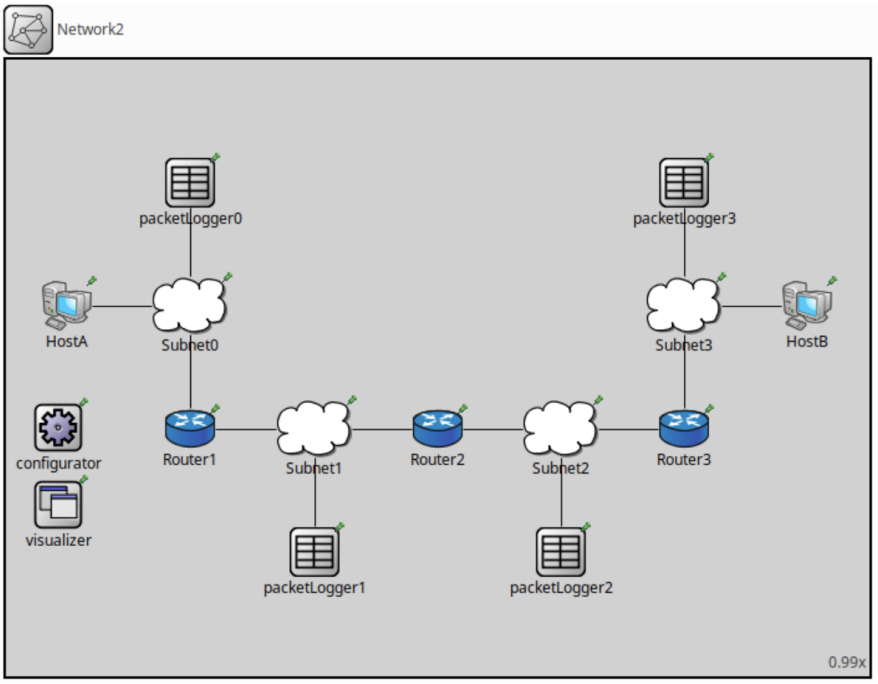
[4]Her er ikke tages højde for loopback interface, lo0.

4.2 Del 8 - Større IP-netværk med statisk konfiguration

I denne del af øvelsen bliver der fejlsøgt på et større statisk konfigureret netværk. Fejlene bliver rettet og dokumenteret. Igen er hvert subnet udstyret med en packerLogger, som viser al trafikken, som passerer subnettet. Et routing loop konstrueres og analyseres.

4.2.1 Del 8a - Fejlsøgning i IP netværk

Dette er delen med fejlsøgning.



Figur 28: Network 2 - Topologi til del 8

Topologien til denne del af øvelsen ses af Figur 28. I simulationen vil HostA gerne sende en ping request til HostB, men lokalnettet er fejlagtigt statisk konfigureret. Dette ses ved at afvikle simulationen og betragte animationerne. HostA får sendt sin request, men modtager aldrig en reply. Netværket var konfigureret på forhånd, så fokus var på at rette fejlene. Ved at fejlfinde på netværket konstateres følgende:

1. Router2 havde i sin routing-table en fejl på Subnet3. Der stod 168.192.3.0/24, som IKKE er en privat IP-adresse. Vi rettede denne værdi i routing table til 192.168.3.0/24, som overholder reglerne for private IP-adresser.
2. Router2 havde en fejl i routing-konfiguration, hvor den havde sat et forkert netværk til et forkert interface. Den var konfigureret til at videresende pakker til 192.168.1.1 (Router1's interface i Subnet0) ud af interface eth1. Dog sad 192.168.1.1 ikke for enden af dette interface, så det blev rettet til eth0 i stedet. Interface eth1 var Router2's interface i Subnet2, som ikke er en gyldig vej til Subnet0 jf. Figur 28

Efter ovenstående rettelser blev simulation afviklet, og der forekom en succesfuld ping-transmission, som ses af Figur 29:

```
No.     Time      Source         Destination          Protocol Length Info
1 0.000000  192.168.0.2          192.168.3.2          ICMP     102    Echo (ping) request  id=0x0000, seq=0/0, ttl=32 (reply in 2)

No.     Time      Source         Destination          Protocol Length Info
2 0.000007  192.168.3.2          192.168.0.2          ICMP     102    Echo (ping) reply    id=0x0000, seq=0/0, ttl=29 (request in 1)
```

Figur 29: packetLogger0 capture data

4.2.2 Del 8b - Routing loops i IP netværk

Som nævnt i øvelsesvejledningen[5], er et kendt problem i IPv4 netværk, de såkaldte routing loops. De forekommer, når en IP-pakke gentagne gange bliver videresendt mellem routere, men aldrig når frem til den tiltænke destination. For at konstruere dette i en simulation, ændredes følgende i routing-table for Router3:

1. Vi tilføjede en linje, der sagde, at alle pakker med destination i Subnet3, skulle videresendes ud af routerens interface, som var tilsluttet Subnet2 i stedet for.
2. Når Router2 igen får pakken, kan den se, at den skal sende ud til Subnet3, og videresender igen til Router3. Herefter opstår et loop, og pakken når aldrig frem. Grunden til at den ikke looper for evigt, er TTL (Time-To-Live), som er et felt i IPv4 headeren, der formindskes med 1 for hvert hop.

```
No.     Time      Source       Destination  Info
      1 0.000000  192.168.0.2  192.168.3.2  Echo (ping) request  id=0x0000, seq=0/0, ttl=30 (no response found!)

No.     Time      Source       Destination  Info
      2 0.000001  192.168.0.2  192.168.3.2  Echo (ping) request  id=0x0000, seq=0/0, ttl=29 (no response found!)

No.     Time      Source       Destination  Info
      3 0.000002  192.168.0.2  192.168.3.2  Echo (ping) request  id=0x0000, seq=0/0, ttl=28 (no response found!)

No.     Time      Source       Destination  Info
      4 0.000003  192.168.0.2  192.168.3.2  Echo (ping) request  id=0x0000, seq=0/0, ttl=27 (no response found!)

...

...

...

No.     Time      Source       Destination. Info
     31 0.000030  192.168.2.1  192.168.0.2  Time-to-live exceeded (Time to live exceeded in transit)
```

Figur 30: loop-trafik indsmalet af packetLogger2

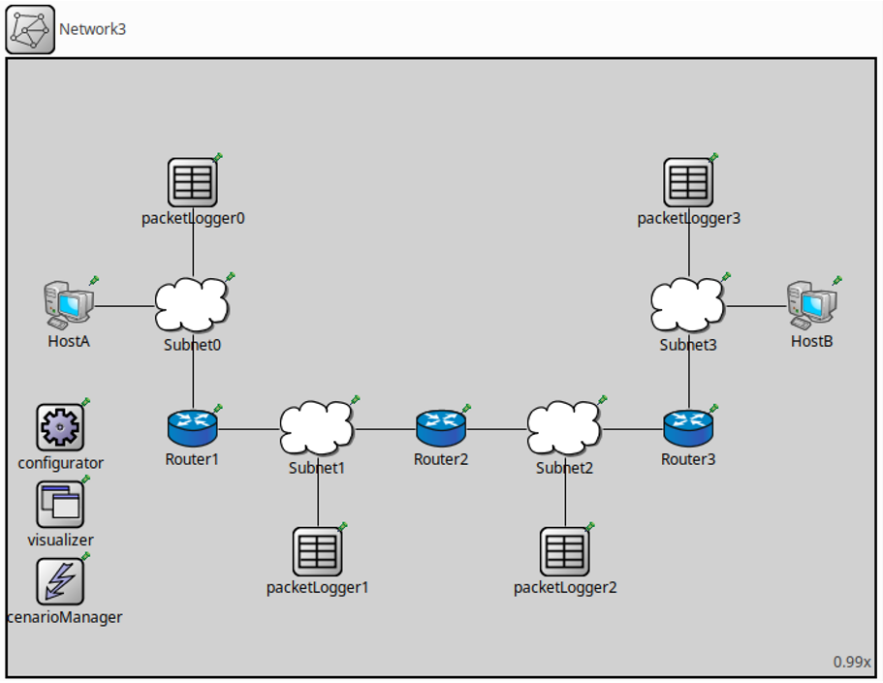
Indholdet i Figur 30 er forkortet en smule, så kun de relevante dele af headeren er taget med. Ved at undersøge trafikken, konstateres det, at hver pakke har en ttl (TTL). Til at starte med er ttl sat til 30. For hver gang pakken optræder i vores capture, falder ttl-værdien med 1. I vores capture, kan vi se, at pakken bliver droppet i frame nr. 31, fordi ttl når ned på 0.

[5] [2], Staalhagen

4.3 Del 9 - Dynamisk routing i IP

4.3.1 Del 9a - RIPv2

I denne del af øvelsen bliver der undersøgt hvordan routere vha. RIPv2[6] kan udveksle informationer om forskellige subnet, som de har kendskab til, og hvordan de indbyrdes etablerer deres egne routing tables. Der undersøges også, hvordan routere håndterer hvis en af deres links bliver utilgængelige.



Figur 31: Network 3 - Topologi til del 9

I starten af del 9 tages der udgangspunkt i det samme netværk som i del 8, dog konfigureret med dynamisk routing. IP-adresserne på de forskellige nodes er stadig statisk konfigureret. Til dynamsik routing bruger routerne RIP til at udveksle informationer om hinandens kendte netværk. Disse informationer bruges til at etablere den hurtigste rute fra end-to-end.

Til at starte med afvikles en simulation, hvor HostA prøver at ping'e HostB. Dette kan ikke lade sig gøre, før vi slår RIP til i netværket. Efter RIP blev slået til i netværket, blev simulationen afviklet, og animationerne viste så småt en succesfuld end-to-end forbindelse.

[6] Routing Information Protocol ver. 2

```
<config>
    <interface hosts='HostA' names='eth0' address='192.168.0.2' netmask='255.255.255.0' />
    <interface hosts='HostB' names='eth0' address='192.168.3.2' netmask='255.255.255.0' />
    <interface hosts='Router1' names='eth0' address='192.168.0.1' netmask='255.255.255.0' />
    <interface hosts='Router1' names='eth1' address='192.168.1.1' netmask='255.255.255.0' />
    <interface hosts='Router2' names='eth0' address='192.168.1.2' netmask='255.255.255.0' />
    <interface hosts='Router2' names='eth1' address='192.168.2.1' netmask='255.255.255.0' />
    <interface hosts='Router3' names='eth0' address='192.168.2.2' netmask='255.255.255.0' />
    <interface hosts='Router3' names='eth1' address='192.168.3.1' netmask='255.255.255.0' />

    <route hosts="HostA" destination="0.0.0.0" netmask="0.0.0.0" gateway="192.168.0.1" interface="eth0" metric="0"/>
    <route hosts="HostB" destination="0.0.0.0" netmask="0.0.0.0" gateway="192.168.3.1" interface="eth0" metric="0"/>
</config>
```

Figur 32: IP-konfiguration til del 9

Som ses af figur 32, er der ikke konfigureret nogen gateways til routerne, hvilket vil sige, at de er nødt til af finde dem vha. RIP. Denne proces undersøges ved at analysere Wireshark capture-data fra packetLogger1 i Subnet1. Dog er det kun de første 10 beskeder, som undersøges.

1	0.000000	192.168.1.1	224.0.0.9	RIPv2	70	Request
2	0.220155	192.168.1.2	224.0.0.9	RIPv2	70	Request
3	0.220156	192.168.1.1	192.168.1.2	RIPv2	90	Response
4	3.631208	192.168.1.2	224.0.0.9	RIPv2	110	Response
5	4.377063	192.168.1.1	224.0.0.9	RIPv2	110	Response
6	8.443195	192.168.1.2	224.0.0.9	RIPv2	70	Response
7	12.832203	192.168.1.1	224.0.0.9	RIPv2	70	Response
8	30.000000	192.168.1.1	224.0.0.9	RIPv2	130	Response
9	30.220155	192.168.1.2	224.0.0.9	RIPv2	130	Response
10	60.000000	192.168.1.1	224.0.0.9	RIPv2	130	Response

Figur 33: packetLogger1 capture data

Af Figur 33 fremgår de første 10 RIP beskeder mellem routerne. Hver værdi i hver række skal læses som: frame-nr i caputre, tidspunkt for opsnapning, source ip, destination ip, protokol, length, og type (Request/Response). Sekvensen starter med, at Router1 sender en RIP request ud til alle routerne på det pågældende subnet (224.0.0.9 er en multicast adresse til routere, som benytter RIPv2)[7], som i dette tilfælde er Subnet1. Hvis den første frame (sendt fra Router1) åbnes, får man følgende RIP output:

```
Routing Information Protocol
Command: Request (1)
Version: RIPv2 (2)
Address not specified, Metric: 16
Address Family: Unspecified (0)
Route Tag: 0
Netmask: 0.0.0.0
Next Hop: 0.0.0.0
Metric: 16
```

Figur 34: Frame nr. 1 fra Figur 33

Efterfølgende til $t \approx 0,220155s$, sender Router2 en tilsvarende RIP request ud, som også har en Metric på 16. En request besked beder alle routere om at sende en Response med indholdet i deres respektive routing-tabel. Da routerne ikke kender nogen netværk endnu, har de første to requests en metric værdi på 16. En metric værdi kan være et heltal mellem 1-15.[8] Værdien

[7] [3]: RFC 2453, afs. 4.5
[8] [4]: RFC 2453, afs. 3.5

angiver, hvor mange "hops"den pågældende router skal bruge for at nå et givent netværk. En værdi på 16 betyder, at netværket ikke kan nås. 16 betyder "infinity".[9]

Umiddelbart efter disse to requests, sender Router1 en besked ud til Router2, med sin nye routing-table. Den har nemlig fået kendskab til Router2 vha. de første requests:

```
Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.2
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
Command: Response (2)
Version: RIPv2 (2)
IP Address: 192.168.0.0, Metric: 1
    Address Family: IP (2)
    Route Tag: 0
    IP Address: 192.168.0.0
    Netmask: 255.255.255.0
    Next Hop: 0.0.0.0
    Metric: 1
IP Address: 192.168.1.0, Metric: 1
    Address Family: IP (2)
    Route Tag: 0
    IP Address: 192.168.1.0
    Netmask: 255.255.255.0
    Next Hop: 0.0.0.0
    Metric: 1
```

Figur 35: Frame nr. 3 fra Figur 33

Efterfølgende i packetLogger1's capture, har Router2 fået en ny indgang i sin routing tabel via. Router1. Router1 kender Subnet0, og sender det videre til Router2. Fordi Router2 har fået denne nye indgang i sin routing-tabel, sender den en RIP response (til 224.0.0.9) ud med sine nye værdier i routing-tabellen:

[9] [5]: RFC 2453, afs. 3.4.1

```
Internet Protocol Version 4, Src: 192.168.1.2, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
Command: Response (2)
Version: RIPv2 (2)
IP Address: 192.168.1.0, Metric: 1
    Address Family: IP (2)
    Route Tag: 0
    IP Address: 192.168.1.0
    Netmask: 255.255.255.0
    Next Hop: 0.0.0.0
    Metric: 1
IP Address: 192.168.2.0, Metric: 1
    Address Family: IP (2)
    Route Tag: 0
    IP Address: 192.168.2.0
    Netmask: 255.255.255.0
    Next Hop: 0.0.0.0
    Metric: 1
IP Address: 192.168.0.0, Metric: 2
    Address Family: IP (2)
    Route Tag: 0
    IP Address: 192.168.0.0
    Netmask: 255.255.255.0
    Next Hop: 0.0.0.0
    Metric: 2
```

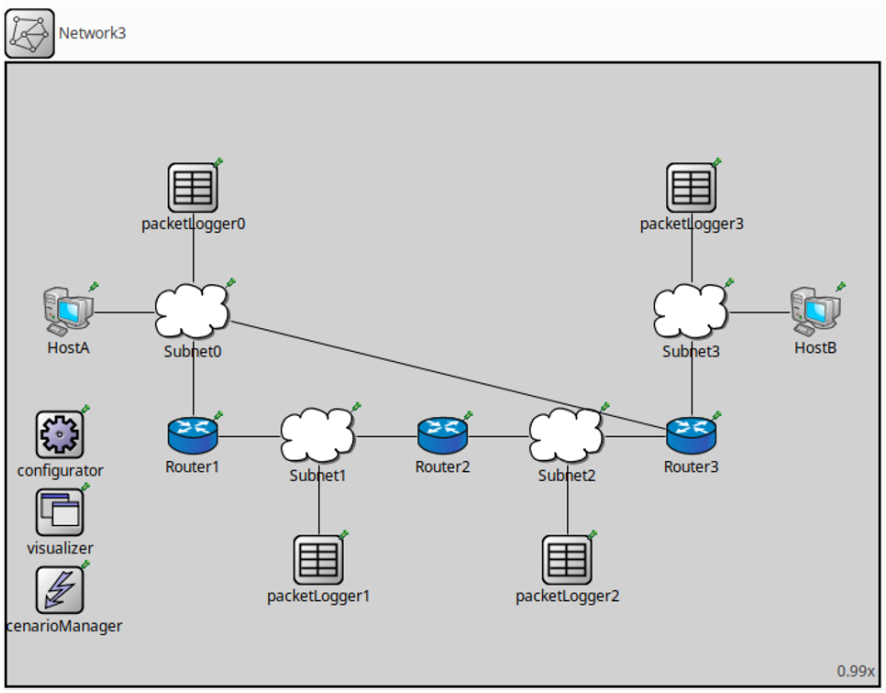
Figur 36: Frame nr. 4 fra Figur 33

Dvs. at Router2 kan nå en vilkårlig destination i Subnet0 med 2 hop, Subnet1 og Subnet2 med hhv 1 hop. Et lignende mønster gentager sig t.o.m frame nr. 7. Når ruterne er blev initialiseret færdigt med RIP, sender routerne periodiske opdateringer ud hvert 30. sekund. Dette er standard procedure for RIPv2.

Til $t = 30s$ i Figur 33 begynder begge routerne at sende periodiske opdateringer ud hvert 30. sekund. Router1 sender periodiske opdateringer ca. $0,22s$ før Router2.

Routerne lærer af hinanden, da de sender hele indholdet af deres routing table ud, som multicast til alle routere. Disse multicast beskeder, bruger de andre rutere, til at udregne metrics til deres egne routing tables.

4.3.2 Del 9b - Samme netværk som i 9a, dog med ny forbindelse



Figur 37: Topologi hvor Router3 har forbindelse i Subnet0

Figur 37 viser topologien for denne del af øvelsen. Router3 har fået et interface forbundet i Subnet0, og har IP-adresse: 192.168.0.4. IP-konfigurationen for det øvrige netværk er stadig den samme som tidligere i øvelsen. Her kunne det være interessant at undersøge hvilken vej, Router1 mener er den bedste, hvis den skal videresende pakker til Subnet3, da Subnet0 har en kortere vej til Subnet3 end før.

Simulationen afvikles, og packerLogger1 opsnapper trafikken, som fremgår af Figur 38

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.1	224.0.0.9	RIPv2	70	Request
2	0.831879	192.168.0.4	224.0.0.9	RIPv2	70	Request
3	0.831880	192.168.0.1	192.168.0.4	RIPv2	90	Response
4	4.377063	192.168.0.1	224.0.0.9	RIPv2	110	Response
5	5.263661	192.168.0.4	224.0.0.9	RIPv2	130	Response
6	9.652669	192.168.0.1	224.0.0.9	RIPv2	70	Response
7	30.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
8	30.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response
9	60.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
10	60.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response

Figur 38: packetLogger0 capture data

Det tages udgangspunkt i frame nr. 9, da routerne her allerede er begyndt at sende periodiske opdateringer ud, dvs. de har initialiseret ruterne.

```
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
      ...
       afkortet
      ...
        IP Address: 192.168.3.0, Metric: 2
            Address Family: IP (2)
            Route Tag: 0
            IP Address: 192.168.3.0
            Netmask: 255.255.255.0
            Next Hop: 0.0.0.0
            Metric: 2
```

Figur 39: Frame nr. 9 tilhørende Figur 38.

Her fås en metric værdi på 2, dvs. at den har beregnet den hurtigste rute til at være 2 hop. Jf. Figur 37, vil vejen til Subnet3 med 2 hop være igennem Subnet0 til router3, og så videre til destinationen. Altså den videresender disse pakker til 192.168.0.4, som er router3's interface i Subnet0.

Af Figur 40 fremgår ping-transmissionen til $t \approx 300s$ Der sendes to ICMP (ping) requests, men kun én reply, da der ikke var noget svar på den første request. Der kommer dog reply efter request nr. 2 (frame nr. 27).

```
    25 297.255932  192.168.0.2    192.168.3.2    ICMP      102 Echo (ping) request  id=0x0003, seq=0/0, ttl=32 (no response found!)
    26 297.255933  192.168.0.2    192.168.3.2    ICMP      102 Echo (ping) request  id=0x0003, seq=0/0, ttl=31 (reply in 27)
    27 297.255936  192.168.3.2    192.168.0.2    ICMP      102 Echo (ping) reply    id=0x0003, seq=0/0, ttl=31 (request in 26)
```

Figur 40: Ping transmission til $t \approx 300s$

Forskellen i tid på frame nr. 25 til nr. 26, er ca. 0.000001s, og der bliver umiddelbart efter lavet en succesfuld ping transmission.

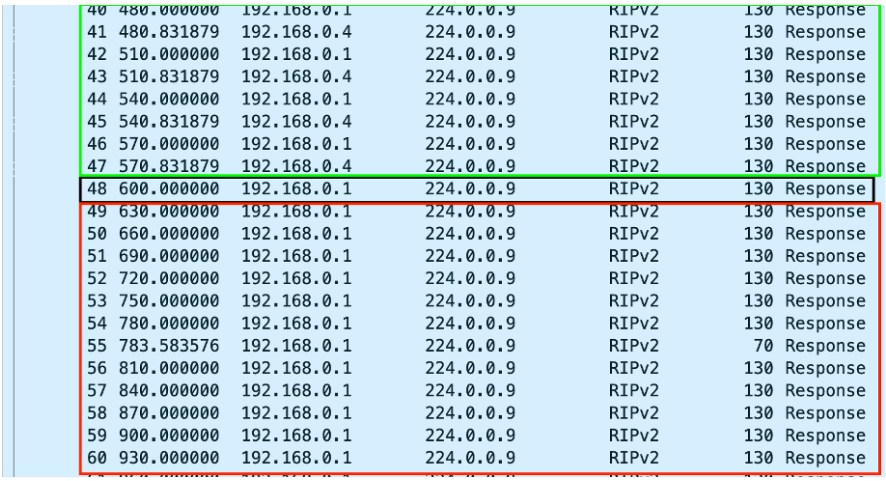
Ping requesten i frame 25, bliver sendt meget kort tid inden de periodiske RIP opdateringer, og derfor kan der muligvis have været noget bearbejdning af rutning, der kunne være skyld i at den første ping request fejler. RIP opdateringerne sker hvert 30. sekund, og derfor sker ping transmission mellem den 9. og 10. opdatering.

Dog har netværket ikke ændret sig inden ping transmissionen fandt sted, så ruterne burde være initialiseret korrekt.

4.3.3 Del 9c - Håndtering af fejl i netværket med RIPv2

Til denne del af øvelsen undersøges hvordan RIP håndterer, hvis en forbindelse svigter undervejs i simulationen. Der tages udgangspunkt i samme topologi som i del 9b, dog hvor forbindelsen mellem Router3 og Subnet0 svigter til $t = 600s$ i simulationen. (topologi fremgår af Figur 37).

Simulationen afvikles, og der analyseres for packetLogger0 capturedata som ses af Figur 41



40	480.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
41	480.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response
42	510.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
43	510.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response
44	540.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
45	540.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response
46	570.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
47	570.831879	192.168.0.4	224.0.0.9	RIPv2	130	Response
48	600.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
49	630.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
50	660.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
51	690.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
52	720.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
53	750.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
54	780.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
55	783.583576	192.168.0.1	224.0.0.9	RIPv2	70	Response
56	810.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
57	840.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
58	870.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
59	900.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response
60	930.000000	192.168.0.1	224.0.0.9	RIPv2	130	Response

Figur 41: packetLogger0 capture data

Frames markeret i den grønne boks er før linket blev fjernet, og pakkerne i den røde boks er efter linket blev fjernet. Her ses det, at pakkerne i den grønne boks har src: skiftevis router1 og router3. I den røde boks er src: kun router1.

Efter linket blev fjernet, sender Router1 responses ud, hvor den siger at den har en metric på 2, til Subnet3. Der gør den flere gange indtil frame 54, hvor den sætter metric til 16, og hvor den ca. 3 sekunder efter i frame 55 sætter metric til 3. At den sætter metric til 16 er en indikation på at den har opdaget fejlen. Lovlige metric værdier var, som tidligere nævnt, et heltal fra 1-15.

Følgende capture data tilhører Figur 41:

```
Frame 53: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits)
Ethernet II, Src: 0a:aa:00:00:00:03 (0a:aa:00:00:00:03), Dst: IPv4mcast_09 (01:00:5e:00:00:09)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
        Command: Response (2)
        Version: RIPv2 (2)
        IP Address: 192.168.0.0, Metric: 1
        IP Address: 192.168.1.0, Metric: 1
        IP Address: 192.168.2.0, Metric: 2
        IP Address: 192.168.3.0, Metric: 2

Frame 54: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits)
Ethernet II, Src: 0a:aa:00:00:00:03 (0a:aa:00:00:00:03), Dst: IPv4mcast_09 (01:00:5e:00:00:09)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
        Command: Response (2)
        Version: RIPv2 (2)
        IP Address: 192.168.0.0, Metric: 1
        IP Address: 192.168.1.0, Metric: 1
        IP Address: 192.168.2.0, Metric: 2
        IP Address: 192.168.3.0, Metric: 16
```

```
Frame 55: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Ethernet II, Src: 0a:aa:00:00:00:03 (0a:aa:00:00:00:03), Dst: IPv4mcast_09 (01:00:5e:00:00:09)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
        Command: Response (2)
        Version: RIPv2 (2)
        IP Address: 192.168.3.0, Metric: 3
```

Det ses også af Figur 41 at længden af frame 55 er 70, som er en indikation på en opdateret værdi i dens routing-tabel. Frame nr. 55 sender kun den nye opdatering ud, men ikke alle de subnets den kender. Det sker først i frame nr. 56:

```
Ex, frame no. 56:
Frame 56: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits)
Ethernet II, Src: 0a:aa:00:00:00:03 (0a:aa:00:00:00:03), Dst: IPv4mcast_09 (01:00:5e:00:00:09)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
        Command: Response (2)
        Version: RIPv2 (2)
        IP Address: 192.168.0.0, Metric: 1
        IP Address: 192.168.1.0, Metric: 1
        IP Address: 192.168.2.0, Metric: 2
        IP Address: 192.168.3.0, Metric: 3
```

Router1 finder den rigtige vej til Subnet3 til tidspunktet mellem frame 54 og 55. Denne capture overvåger Subnet0, men Router1 er også forbundet til Subnet1 jf. topologien. Den har fået den korrekte rute fra router2 i Subnet1, hvilket ikke fremgår af dette capture, fordi packetLogger0 ikke er forbundet til Subnet1, og den har efterfølgende rettet fejlen, samt sendt responses ud med rettelsen.

5 Konklusion

I denne rapport har vi arbejdet med og undersøgt forskellige Point-to-point protokoller med hhv. Stop-and-Wait og Sliding-Window. Vi har undersøgt hvordan forskellige OSI lag2 enheder performer i forhold til hinanden, og hvordan IP-protokollen fungerer på OSI lag3.

Empiri:

1. Det er undersøgt, hvad udnyttelsen af en forbindelse er, samt dens sammenmhæng med thoughputtet hos en modtager. Derudover hvordan transmisionsfejl påvirker dem.
2. Tilsvarende er der brugt sliding window til at undersøger, hvordan udnyttelsen stiger ved et større windowsize. Hvorefter ARQ-principperne, Go-back-N og Selective-Reject's, fordele og ulemper er undersøgt, ved transmissionsfejl.
3. Et IP-netværk blev statisk konfigureret, og der blev opnået succesfulde end-to-end forbindelser. Det gjorde os i stand til at fejlsøge på et andet netværk, som var konfigureret forkert og dermed udbedre fejlene.
4. Forsinkelsen af transmissionerne over et hub-baseret LAN er blevet undersøgt og der er blevet forklaret, hvorfor end-to-end forsinkelsen var konstant når payload size var under 43 bytes. Desuden er det blevet forklaret, hvorfor at man ønsker en minimumsstørrelse på 46 bytes i payloaden i et kablet netværk hvor CSMA/CD benyttes.
5. Forskellen på forsinkelser over LAN og WLAN er blevet undersøgt og dermed er forskellen mellem collision detection og collision avoidance forklaret samt diskuteret og konkluderet hver deres fordele og ulemper.
6. Forskellen på den egentlige bitrate og throughput er blevet undersøgt og diskuteret samt konkluderet, hvorfor bytes modtaget som funktion af bitrate ikke er konstant og hvorfor dette skyldes.
7. Konstrueret et routing loop, og undersøgt hvordan Time-To-Live feltet i IPv4 headeren forhindrer at en vildfaren IP-pakke looper for evigt i et netværk.
8. Der blev undersøgt hvordan routere kan benytte RIP ver. 2, til initalisere ruter mellem forskellige subnet, og hvordan de ved hjælp af feltet Metric beslutter den hurtigste rute fra A til B, når der er flere ruter at vælge imellem. Efter initialisering af ruterne, blev der sendt periodiske opdateringer ud, for at holde de andre routere ajour med ruterne.
9. Netværkstraffik blev analyseret for at se hvordan RIP virker, når et link går ned (Point to point forbindelse går tabt). Vha. periodiske opdateringer kan routerne opdage hvis en rute bliver utilgængelig. Periodiske opdateringer sendes ud hvert 30. sekund.

Litteratur

[1] Staalhagen, L., "Introduktion til datanet", 2024

[2] Staalhagen, L.,"Øvelsesvejledning - Del 7-9", 2024

[3] RFC 2453, RIP Verson 2 - Internet Standard, afs. 4.5: Multicasting: `https://www.rfc-editor.org/rfc/rfc2453.html#section-4.5`, 1998

[4] RFC 2453, RIP Verson 2 - Internet Standard, afs. 3.5: Protocol specification: `https://www.rfc-editor.org/rfc/rfc2453.html#section-3.5`, 1998

[5] RFC 2453, RIP Verson 2 - Internet Standard, afs. 3.4.1: Dealing with changes in topology: `https://www.rfc-editor.org/rfc/rfc2453.html#section-3.4.1`